Under consideration for publication in Math. Struct. in Comp. Science

Deriving Session and Union Types for Objects[†]

Lorenzo Bettini[1], Sara Capecchi[1], Mariangiola Dezani-Ciancaglini[1], Elena Giachino[2], Betti Venneri[3]

[1]*(dezani@di.unito.it) Dipartimento di Informatica, Università di Torino, corso Svizzera 185, 10131 Torino*
[2]*Focus Research Team, Università di Bologna/INRIA, mura Anteo Zamboni 7, 40127 Bologna*
[3]*Dipartimento di Sistemi e Informatica, Università di Firenze, viale Morgagni 65, 50134 Firenze*

Received 20 December 2010; Revised 24 September 2011

Guaranteeing that the parties of a network application respect a given protocol is a crucial issue. *Session types* offer a method for abstracting and validating structured communication sequences (sessions). *Object-oriented programming* is an established paradigm for large scale applications. *Union types*, which behave as the least common supertypes of a set of classes, allow implementing unrelated classes with similar interfaces without additional programming. We have previously developed an integration of the features above into a class-based core language for building network applications, which successfully amalgamates sessions and methods in order to flexibly exchange data according to communication protocols (session types).
The first aim of this work is to provide a full proof of the type safety property for that core language, by renewing syntax, typing and semantics. Hence static typechecking guarantees that, after a session has started, computation cannot get stuck on a communication deadlock.
The second aim is to define a constraint-based type system which reconstructs the appropriate session types of session declarations, instead of assuming that session types are explicitly given by the programmer; such algorithm can save programming work, and automatically presents an abstract view of the communications of the sessions.

Keywords: Sessions, Object Oriented Programming, Session Types, Union Types.

1. Introduction

When developing network applications it is crucial to have a linguistic mechanism to write safe communication protocols. The current mainstream programming languages, such as, e.g., Java, still leave to the programmer most of the responsibility in guaranteeing that the communication will evolve as agreed by all the involved agents. The standard type systems can only provide a way of declaring the types of the exchanged data, but they cannot guarantee that a communication protocol is respected, thus avoiding that a client-server application gets stuck because of an error in the communication sequence.

Session types (Honda, 1993; Honda et al., 1998) were introduced as a mechanism for abstracting structured communication sequences (*sessions*) and for validating communication protocols.

[†] This work has been partially supported by MIUR Projects DISCO - Distribution, Interaction, Specification, Composition for Object Systems, and IPODS - Interacting Processes in Open-ended Distributed Systems, and by EU Collaborative project n. 257414 ASCENS - Autonomic Service-Component Ensembles.

In this approach communication channels are given types representing the values sent or received. For instance, the type `?int.!bool` expresses that an integer will be received and then a boolean value will be sent (as it is usual in process calculi, `?` and `!` are used for input and output, respectively). A session, in order to respect a communication protocol, must involve channels of dual session types, thus guaranteeing that, after a session has started, the values sent and received will be of the appropriate type and the communication will not get stuck. For instance, if one channel has the above type `?int.!bool`, the other one must have the dual type `!int.?bool`. Since the specification of a session is a type, the conformance test of programs with respect to specifications becomes type checking.

Furthermore, in network applications, it is important to rely on type safe flexibility of exchanged data; thus, we need a mechanism to abstract over the actual types that are communicated over a network protocol. This is even more crucial when execution paths are chosen according to the run-time type of the exchanged objects. For this reason, it seems natural to try to merge communication mechanisms into the popular *object-oriented* programming paradigm; in mainstream object-oriented class-based programming languages writing network communication programs typically involves relying on specific libraries, but these languages do not provide linguistic constructs to directly deal with communications and protocols. Instead, we would like to write class definitions which naturally include communication primitives. With this respect, an amalgamation of communication centred and object-oriented programming has been first proposed in (Drossopoulou et al., 2007), where methods are unified with sessions and choices are based on the classes of exchanged objects.

In an object-oriented class-based context, reusability is based both on subclassing and on the substitutability implied by subtyping, which coincides with subclassing (or interface implementation) in Java-like languages. Thus, this form of reuse must be designed from the start, choosing the right base classes or interfaces, since, although two classes may share some features (methods and fields), if they do not belong to the same hierarchy, their reuse will require refactoring of existing code. A solution to deal with these problems is provided by *union types*, which represent the set unions of objects of several types: a union type behaves as the least common supertype of a set of objects, without requiring to write a specific base class or interface. With union types, in an object-oriented programming scenario, developing independent classes with similar interfaces requires no additional programming (Igarashi and Nagira, 2007).

For these reasons, union types seem to be very useful when communications involve data exchange in the shape of objects as class instances: we can express communications between parties which manipulate heterogeneous objects just by sending and receiving objects which belong to subclasses of one of the classes in the union type. For instance, consider a communication between a bank and a client: the bank can answer `yes` or `no` to a client request, according to the account balance. If `yes` and `no` are objects of classes `OK` and `NoMoney`, respectively, then the class of the object `answer` is naturally the union of the two classes `OK` and `NoMoney`, i.e. $\texttt{OK} \vee \texttt{NoMoney}$. Without union types typing `answer` would require a superclass of both `OK` and `NoMoney` to be already defined; besides manual programming, and possible code refactoring, this superclass could also include unwanted objects. This does not happen with a union type (least common supertype). In this way the flexibility of object-oriented depth-subtyping is enhanced, by strongly improving the expressiveness of choices based on the classes of sent/received objects.

In this paper we merge union types in the amalgamation of sessions and methods, in order to

enhance the network communications of class-based programs relying on session types: in (Bettini et al., 2008a) we presented $\mathscr{F}\mathsf{SAM}^{\vee}$ (*F*eatherweight *S*essions *A*malgamated with *M*ethods plus union types) which formalises the use of union types for session-centred communications in a core object-oriented calculus. $\mathscr{F}\mathsf{SAM}^{\vee}$, as the language of (Drossopoulou et al., 2007), is agnostic w.r.t. to the remaining aspects of the language, such as whether the language is distributed or concurrent, and the features for synchronisation. In $\mathscr{F}\mathsf{SAM}^{\vee}$, sessions are defined in a class (which can have also fields). Sessions and methods are "amalgamated": invocation is made on an object and the execution takes place immediately and concurrently with the requesting thread (indeed, $\mathscr{F}\mathsf{SAM}^{\vee}$ is multi-threaded and the communication is asynchronous). Thus, it keeps the method-like invocation mechanism while involving two threads, typical of session based communication mechanisms. Just like dynamic binding of object-oriented method invocation, the body is determined by the class of the receiving object (avoiding in this way the usual branch/select primitives (Honda et al., 1998)), and any number of communications interleaved with computation is possible. We believe that the above amalgamated model of session naturally reflects our intuition of services. Furthermore, it can neatly encode "standard" methods.

This paper is an extension of (Bettini et al., 2008a) in many respects. First of all, the syntax (and consequently the typing and semantics) was slightly modified. Second, we present the full formalisation of $\mathscr{F}\mathsf{SAM}^{\vee}$, together with the proofs of the type soundness property (from a technical point of view, the amalgamation of union types and session-centred communications poses specific problems in formulating reduction and typing rules to ensure that communications are safe while flexible). Finally, we also introduce a type inference system for the session types of the sessions in classes. In particular, while the type system derives session types for expressions assuming that all session declarations are decorated with explicit session types (and expressions can have many types due to the presence of subsumption), the inference algorithm gives an expression its minimal type and calculates the constraints that must be satisfied in order to reconstruct the related session type (which will be proved unique).

With the type inference system, the programmer is no longer responsible for declaring the session types. Therefore, this inference has a pragmatic motivation, since, due to their "behavioural" nature, session types might tend to be quite long to write, when the communication protocol is not a simple one (especially when recursive types are involved). Thus, having a type inference system for session types can save some programming work, and automatically presents an abstract view of the communications of the sessions. However, in an implementation of our approach, the inference algorithm might not necessarily prevent the programmer from writing session types. For instance, the programmer might decide to write the session types explicitly, and use the inference system as a tool for verifying the written protocols. Alternatively, the inference system might insert the inferred types in the text of the program, so that the programmer can have the abstract view of the protocol, and verify that the protocol is as it was intended. Finally, a mixed approach can be employed: the programmer can write the explicit session types for at least one side of the protocol, and have the type inference system generate the session type for the other part. Summarising, in an implementation, the session type inference system does not necessarily impose to remove all the session type declarations from a program, but it is meant as a tool which should help the programmer while designing and implementing communication protocols. The aim of the presentation of the type inference system in this paper is only to study its theory and properties: how it will be employed by a language designer is out of our scope.

(type)	T	::=	C | T∨T
(class)	L	::=	class C ◁ D { $\overline{\texttt{T f}}$; $\overline{\texttt{S}}$ }
(session)	S	::=	T t s{ e }
(expression)	e	::=	x | this | $\texttt{cont}^{\texttt{T}}$ | o | e; e | e.f:= e | e.f | new C($\overline{\texttt{e}}$)
		|	e.s{e} | e•s{ k }
		|	k. sendC(e){$\texttt{C}_1 \Rightarrow \texttt{e} [] \texttt{C}_2 \Rightarrow \texttt{e}$}
		|	k. recvC(x){$\texttt{C}_1 \Rightarrow \texttt{e} [] \texttt{C}_2 \Rightarrow \texttt{e}$}
		|	k. sendW(e){$\texttt{C}_1 \Rightarrow \texttt{e} [] \texttt{C}_2 \Rightarrow \texttt{e}$}
		|	k. recvW(x){$\texttt{C}_1 \Rightarrow \texttt{e} [] \texttt{C}_2 \Rightarrow \texttt{e}$}
(parallel threads)	*P*	::=	e | *P* || *P*
(session type)	t	::=	ε | t.t | α | †{$\texttt{C}_1 \Rightarrow \texttt{t} [] \texttt{C}_2 \Rightarrow \texttt{t}$} | $\mu\alpha$.†{$\texttt{C}_1 \Rightarrow \texttt{t} [] \texttt{C}_2 \Rightarrow \texttt{t}$} | ◎

Fig. 1. Syntax, where syntax occurring only at run time appears shaded.

Paper Structure. The description of the calculus $\mathscr{F}\mathsf{SAM}^{\vee}$ with its operational semantics is given in Sections 2, 3 and 4. Section 5 presents the type system, whose properties are in Section 6. Sections 7 and 8 are devoted to the type inference system. Finally, in Sections 9 and 10 we discuss related works and draw some conclusions.

2. The calculus: $\mathscr{F}\mathsf{SAM}^{\vee}$

This section presents the syntax of $\mathscr{F}\mathsf{SAM}^{\vee}$ (Figure 1), a minimal concurrent and imperative core calculus, based on *Featherweight Java* (Igarashi et al., 2001) (abbreviated with FJ). $\mathscr{F}\mathsf{SAM}^{\vee}$ supports the basic object-oriented features and session request, a form of session delegation, branching sending/receiving and loops.

In Figure 1 we use grey to indicate *run-time expressions*, that are produced during the reduction process and do not occur in the *user expressions*. We use the standard convention of denoting with $\overline{\xi}$ a sequence of elements $\xi_1, ..., \xi_n$.

Types, ranged over by T, are defined as in (Igarashi and Nagira, 2007): they are built out of class names by the union operator (denoted by $\vee$).

Programs are defined from a collection of classes. The metavariables C and D, possibly with subscripts, range over class names. Each class has a suite of *fields* of the form $\overline{\texttt{T f}}$, where f represents the field name and T its type, and a suite of *session declarations* $\overline{\texttt{S}}$. As in FJ, the fields declared by a class are added to the ones of the superclass and the resulting sequence of fields is assumed to contain no duplicate names. We declare sessions just like as we declare methods in Java classes, with the new remarkable feature that their bodies can include communication operations. Since sessions can encode methods, as we shall see at the end of this section, for simplicity we omit standard methods in our classes. Session declarations are of the form T t s{ e }, where s is the session name, e the session body, T the return type, and t is the session type which describes the communication protocol in the way standard method types describe the protocols for method-call interactions. For the sake of conciseness the symbol ◁ represents class extension, as in (Igarashi et al., 2001). The class Object is implicitly defined in every program; it has no fields and no sessions. A class definition always includes the superclass (even when it is Object).

Expressions include variables, that are both standard term variables $\mathtt{x}$ and the special variables $\mathtt{this}$ and $\mathtt{cont}^{\mathtt{T}}$. The variable $\mathtt{this}$ is considered implicitly bound in any session declaration. $\mathtt{sendW}$ and $\mathtt{recvW}$ are the only binders for the free occurrences of $\mathtt{cont}^{\mathtt{T}}$ inside their bodies, where $\mathtt{cont}^{\mathtt{T}}$ represents the continuation by recursive computation. The intuition is that $\mathtt{sendW}$ and $\mathtt{recvW}$ expressions will be unfolded, when necessary, during evaluation by replacing the free occurrences of $\mathtt{cont}^{\mathtt{T}}$ in their bodies with the whole expressions. Note that, for any type $\mathtt{T}$, a special variable $\mathtt{cont}^{\mathtt{T}}$ is provided: it is decorated by the type $\mathtt{T}$ in order to represent the recursive computation of an expression of type $\mathtt{T}$.

As usual, an expression is *closed* if it does not contain free variables.

Object identifiers, denoted by $\mathtt{o}$, are generated at run time when creating objects (by $\mathtt{new}$ expressions).

The expression $\mathtt{e.s}\{\mathtt{e}'\}$ is a *session request* where $\mathtt{e}'$ is called the *co-body* of the request; by operational rules, $\mathtt{e}$ is evaluated to an object $\mathtt{o}$, and the session body of $\mathtt{s}$ in $\mathtt{o}$'s class is executed concurrently with $\mathtt{e}'$, by introducing a new pair of fresh channels, $\mathtt{k}$ and the dual $\tilde{\mathtt{k}}$ (one for each communication direction), to perform communications between the session body and the co-body. Then the evaluation of session requests has a crucial effect on the syntax: it generates parallel threads and introduces communication channels (which are implicit in the source language).

The expression $\mathtt{e} \bullet \mathtt{s}\{\mathtt{k}\}$ represents the *session delegation* in the sense that the execution of the session $\mathtt{s}$ is delegated to the object resulting from the evaluation of $\mathtt{e}$. This means that the current object in order to safely continue the communication with its partner needs to borrow a capability from another object. In this sense we kept the term "delegation" usually encountered in the session types literature. Our notion of delegation diverges slightly from the standard one. In our case the current object asks another object to provide a functionality in its place, without releasing the control of the session: the session channel is not moved around and the current thread executes the code of the delegated object. Technically this is very close to the standard method invocation. The standard session delegation, on the contrary, requires that a private channel is sent to another thread that will be taking care of the session communication on the received channel, while the current thread is excluded from the session. This higher order use of channels cannot be easily expressed in our setting where channels are only created at run time. The channel $\mathtt{k}$ corresponds to the subject of communication expressions inside the session body. We refer to Section 4 (in particular the explanation of reduction rule SESSDEL-R) for further details.

The body of a *communication expression* is a pair of alternatives $\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}$, whose choice depends on the class of the object that is sent or received. The expression $\mathtt{sendC}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}$ evaluates $\mathtt{e}$ to an object and sends it on the active channel, and then continues with $\mathtt{e}_i$, where $\mathtt{C}_i$ is the class that best fits the class of the object sent (if $\mathtt{C}_1 = \mathtt{C}_2$, then the whole expression evaluates to $\mathtt{e}_1$). The counter part of $\mathtt{sendC}$ is the expression $\mathtt{recvC}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}$, where the choice is based on the class of the object received. The expression $\mathtt{sendW}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}$ (where W means While) is similar to $\mathtt{sendC}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}$, except that it allows for enclosed $\mathtt{cont}^{\mathtt{T}}$, which continues the execution at the nearest enclosing $\mathtt{sendW}$. The expression $\mathtt{recvW}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}$ has the obvious meaning.

Note that recursion on objects (via $\mathtt{this}$) is not suitable in our setting for expressing cycles inside single sessions, since it would give rise to different sessions.

Parallel threads, ranged over by P, are run-time expressions or parallel compositions of run-time expressions, where a run-time expression is either a user expression (i.e. an expression in Figure 1 without shaded syntax) or an expression containing channels and/or object identifiers.

In *session types* we use † as a symbol that stands for either ! or ?. By ε we denote the *empty* communication, and the *concatenation* $\mathtt{t}_1.\mathtt{t}_2$ expresses the communications in $\mathtt{t}_1$ followed by those in $\mathtt{t}_2$. Concatenation of session types is used for typing sequential composition of expressions, see rule SEQ-T in Figure 9. The session type ε is the neutral element of concatenation, so that $\varepsilon.\mathtt{t} = \mathtt{t} = \mathtt{t}.\varepsilon$ for all $\mathtt{t}$.

The types $!\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}$ and $?\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}$ express the sending and the receiving of an object, respectively: depending on the class $\mathtt{C}_i$ of this object the communication will proceed with the one of type $\mathtt{t}_i$. In $\mu\alpha.\dagger\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}$ the *session type variable* α can occur inside $\mathtt{t}_i$ with the usual meaning of representing the whole session type. We consider recursive session types modulo fold/unfold: i.e., $\mu\,\alpha.\mathtt{t} = [\mu\,\alpha.\mathtt{t}/\alpha]\mathtt{t}$. So we equate $\mu\alpha.\dagger\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}$ to $\dagger\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}$ when α does not occur in $\dagger\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}$.

The type $\circledcirc$ is used only as session type for $\mathtt{cont}^{\mathtt{T}}$: it plays the role of a place holder which will be replaced by a type variable when the while expression is completed (see rules SENDW-T and RECEIVEW-T in Figure 9).

The following example shows the expressiveness of $\mathscr{F}\mathsf{SAM}^{\vee}$ in a typical collaboration pattern. We refer to (Bettini et al., 2008a) for further motivating examples of our language constructs.

Example 2.1. The interaction we show is between a calculator and a client (Figures 2 and 3). The `Client` sends integer values which are summed by the `Calculator`; this interaction iterates until the `Client` sends a character to notify that the addends are over. Then the `Client` sends to the `Calculator` an object indicating the display-mode of the result (`Paper` or `Video`); finally the `Calculator` displays the result.

The session types `Sum_ST` and `Print_ST` (Figure 2) describes the protocol from the point of view of the `Calculator`. The recursive type

$$\mu\alpha.\ ?\{\mathtt{Int} \Rightarrow \alpha, \mathtt{Char} \Rightarrow ?\{\mathtt{Paper} \Rightarrow \varepsilon,\ \mathtt{Video} \Rightarrow \varepsilon\}\}$$

describes the `Calculator` getting the addends from the `Client`. The first branch represents the case in which the `Calculator` receives an integer from the `Client`; in this case the iteration goes on, i.e. the `Calculator` receives the next input. The second branch represents the case in which the `Client` sends to the `Calculator` a character to signal that addends are over: in this case a further object is expected indicating the mode in which the result must be displayed. In the two branches $\{\mathtt{Paper} \Rightarrow \varepsilon, \mathtt{Video} \Rightarrow \varepsilon\}$ there is no further communications (the session type is ε), since the only action is the print (or display) of the result.

In Figure 2 we find the implementation of the class `Calculator`. It has the field `value` used to store the sum of the addends. The class supports two sessions called `sum` and `print`. The session `sum` has session type `Sum_ST` and return type `Video`$\vee$`Paper`: this union type represents the possible results of the session, i.e. the display modes of the result of the sum. Notice that the return type of the session represents the type of the session body (exactly as return types in

```
1  sessiontype Sum_ST = μα.?{Int ⇒ α, Char ⇒ ?{Paper ⇒ ε, Video ⇒ ε} }
2
3  sessiontype Print_ST = ?{Paper ⇒ ε, Video ⇒ ε}
4
5  class Calculator{
6    Int value;
7    Video∨Paper Sum_ST sum{
8      recvW(x){
9        Int ⇒ value:=value+x; cont^{Video∨Paper};
10       []
11       Char ⇒ this•print;
12     }
13   }
14   Video∨Paper Print_ST print{
15     recvC(y){
16       Paper ⇒ ...; new Paper(); // print the result on paper
17       []
18       Video ⇒ ...; new Video(); // print the result on the screen
19     }
20   }
21 }
```

Fig. 2. The class Calculator.

standard object-oriented languages). Indeed, it is used for dealing with session delegation, when the body of the session is incorporated in the current execution. In this case, we know that the execution of the session body of `sum` will reduce to a value of type `Video∨Paper`. The session type, on the contrary, is needed to deal with session invocation and to check the correctness of the communication. In this case we see that an invocation of the session `sum` must perform a dual communication w.r.t. its session type `Sum_ST`. The session `print` has session type `Print_ST` and return type again `Video∨Paper`. In the body of `sum` the `Calculator` receives an object (line 8) which can be i) of type `Int`, in which case it will be summed to `value` and then the recursion will continue ($\texttt{cont}^{\texttt{Video}\vee\texttt{Paper}}$) or ii) of type `Char`. In the second case the remaining part of the session is delegated to the `Calculator` itself which goes on with session `print` (line 10). The body of the session `print` begins receiving an object indicating the display mode (line 15): according to the class of the received object the field `value` will be printed on paper or displayed on the video.

The session type `Request_ST` (Figure 3) describes the protocol from the point of view of the `Client`. The recursive type

$$\mu\alpha.\,!\{\texttt{Int} \Rightarrow \alpha,\ \texttt{Char} \Rightarrow !\{\texttt{Paper} \Rightarrow \varepsilon,\ \texttt{Video} \Rightarrow \varepsilon\}\}$$

describes the `Client` sending the addends to the `Calculator`: the first branch represents the case in which the `Client` sends an integer to the `Calculator`; in this case the iteration goes on updating and sending the next message; the second branch represents the case in which the `Client` sends to the `Calculator` a character to signal that addends are over: in this case a

```
1  sessiontype Request_ST = μα.!{Int ⇒ α, Char ⇒ !{Paper ⇒ ε, Video ⇒ ε} }
2
3  class Client{
4     Int∨Char msg;
5     Paper∨Video mode;
6     Calculator c;
7     ...
8
9     c.sum{
10       sendW(msg){
11          Int ⇒ update(msg); cont^{Video∨Paper}; // update the content of msg
12          []
13          Char ⇒ sendC(mode){
14              Paper ⇒ ...;
15              []
16              Video ⇒ ...;
17          }
18       }
19    }
20    ...
21 }
```

Fig. 3. The class Client.

further object is sent indicating in which mode the result must be displayed. In Figure 3 we find the implementation of the class `Client`. It has a field of type `Calculator` and two fields `msg` and `mode` used to store the values sent to the `Calculator`. Their types, $\texttt{Int} \vee \texttt{Char}$ and $\texttt{Paper} \vee \texttt{Video}$, respectively, describe the possible classes of the sent values. At line 9 we find an example of session invocation: the `Client` invokes on the `Calculator` `c` the session `sum`. The body of the session invocation (lines 10-16) has session type `Request_ST`. It will be executed in parallel with the body of the session `sum` in the class `Calculator`. Notice that the class `Client` is not fully specified, we just show the code of the session invocation, that must appear somewhere inside a session declaration of the `Client`.

Clearly this example would no longer be typeable if we replaced `Char` by another type, say `Bool`, in the code of the `Client`.

In $\mathscr{F}\mathsf{SAM}^{\vee}$ we adopt some simplifications. First of all, unary choices and n-ary choices are omitted since they can be simply encoded with binary choices (as shown in (Bettini et al., 2008a)). Moreover, types used for selecting branches in a choice are required to be class names, instead of union types. This is not a limitation, since for instance, $\{\mathsf{C}_1 \vee \mathsf{C}_2 \Rightarrow \mathsf{e} \,[]\, \mathsf{C}_3 \Rightarrow \mathsf{e}'\}$ can be encoded as $\{\mathsf{C}_1 \Rightarrow \mathsf{e} \,[]\, \mathsf{C}_2 \Rightarrow \mathsf{e} \,[]\, \mathsf{C}_3 \Rightarrow \mathsf{e}'\}$.

With respect to FJ, in $\mathscr{F}\mathsf{SAM}^{\vee}$ we do not have cast and overriding, which are orthogonal to our issues. We do not have explicit constructors, then in the object instantiation expression $\texttt{new C}(\overline{\mathsf{e}})$, the values $\overline{\mathsf{o}}$ to which $\overline{\mathsf{e}}$ reduce are the initial values of the fields. Furthermore, we omit standard methods since they are seen as special cases of sessions. In fact, a method declaration can be

$$\mathtt{T} <: \mathtt{T} \qquad \frac{\mathtt{T} <: \mathtt{T}' \qquad \mathtt{T}' <: \mathtt{T}''}{\mathtt{T} <: \mathtt{T}''} \qquad \frac{\mathtt{class\ C} \lhd \mathtt{D}\ \{\ \overline{\mathtt{T}\,\mathtt{f}};\ \overline{\mathtt{S}}\ \}}{\mathtt{C} <: \mathtt{D}}$$

$$\mathtt{T} <: \mathtt{T} \vee \mathtt{T}' \qquad \mathtt{T}' <: \mathtt{T} \vee \mathtt{T}' \qquad \frac{\mathtt{T}' <: \mathtt{T} \quad \mathtt{T}'' <: \mathtt{T}}{\mathtt{T}' \vee \mathtt{T}'' <: \mathtt{T}}$$

Fig. 4. Subtyping.

encoded as a session with nested `recvC`s (one for each parameter) and with one `sendC` returning the method body. Similarly, method calls are special cases of session requests: the passing of arguments is encoded as nested `sendC`s (one for each argument) and the object returned by the method body is retrieved via one `recvC`.

3. Auxiliary Functions

As in FJ, a class table CT is a mapping from class names to class declarations with domain $\mathsf{dom}(CT)$. Then a program is a pair $(CT, \mathtt{e})$ of a class table (containing all the class definitions of the program) and an expression `e` (an expression belonging to the source language representing the program's main entry point). The class `Object` does not appear in CT. We assume a fixed CT that satisfies some usual sanity conditions as in FJ (Igarashi et al., 2001). Thus, in the following, instead of writing $CT(\mathtt{C}) = \mathtt{class}\ \ldots$ we will simply write $\mathtt{class\ C}\ \ldots$.

From any CT we can read off the subtype relation between classes, as the transitive closure of $\lhd$ clause; moreover, subtyping is extended to union types as in Figure 4. As usual considering union types modulo the equivalence relation induced by $<:$ we get the commutativity and associativity of $\vee$. Therefore each union type can be written as $\mathtt{C}_1 \vee \ldots \vee \mathtt{C}_n$ for $n \geq 1$: we say that the classes $\mathtt{C}_1, \ldots, \mathtt{C}_n$ *build* the union type $\mathtt{C}_1 \vee \ldots \vee \mathtt{C}_n$. A union type $\mathtt{C}_1 \vee \ldots \vee \mathtt{C}_n$ is *proper* if $n > 1$.

We define auxiliary functions (see Figure 5) to lookup fields and sessions from CT; these functions are used in the typing rules and in the operational semantics. As in FJ these functions may have to inspect the class hierarchy in case the required element is not present in the current class. The difference is that all these functions - but function *sbody* - take a type as argument (not simply a class name) because the receiver expression of a field/session access may be of a proper union type.

As for *field type* lookup, we distinguish between the contexts where the field is used for reading ($ftype_r$) from those where it is used for writing ($ftype_w$). When the field is used in read mode, in case of a proper union type, we simply return the union type of the result of $ftype_r$ invoked on the argument types (if both retrievals succeed). On the contrary, when a field is updated, due to the contravariance relation, in case of a proper union type we must return the intersection of the results of $ftype_w$ on the arguments. However, in the absence of multiple inheritance, either the results are related by subtyping, that is the intersection is exactly one of the classes, or they are not related at all, that is the intersection is empty, thus we can avoid introducing intersection types. For example if objects of class $\mathtt{C}_i$ have a field `f` of class $\mathtt{D}_i$ for $i \in \{1,2\}$ with $\mathtt{D}_1 <: \mathtt{D}_2$, then $ftype_r(\mathtt{C}_1 \vee \mathtt{C}_2) = \mathtt{D}_1 \vee \mathtt{D}_2$ and $ftype_w(\mathtt{C}_1 \vee \mathtt{C}_2) = \mathtt{D}_1$. Instead if $\mathtt{D}_1$ and $\mathtt{D}_2$ are unrelated we get again $ftype_r(\mathtt{C}_1 \vee \mathtt{C}_2) = \mathtt{D}_1 \vee \mathtt{D}_2$, but $ftype_w(\mathtt{C}_1 \vee \mathtt{C}_2)$ is undefined.

The functions *stype* and *rtype* return a set of session types and the return type of a session,

$$fields(\texttt{Object}) = \bullet \qquad \frac{fields(\texttt{D}) = \overline{\texttt{T}'\,\texttt{f}'} \qquad \texttt{class C} \lhd \texttt{D}\ \{\ \overline{\texttt{T}\,\texttt{f}};\ \overline{\texttt{S}}\ \}}{fields(\texttt{C}) = \overline{\texttt{T}\,\texttt{f}}, \overline{\texttt{T}'\,\texttt{f}'}}$$

$$\frac{fields(\texttt{C}) = \overline{\texttt{T}\,\texttt{f}}}{ftype_w(\texttt{f}_i, \texttt{C}) = ftype_r(\texttt{f}_i, \texttt{C}) = \texttt{T}_i}$$

$$ftype_r(\texttt{f}, \texttt{T}_1 \vee \texttt{T}_2) = ftype_r(\texttt{f}, \texttt{T}_1) \vee ftype_r(\texttt{f}, \texttt{T}_2)$$

$$\frac{ftype_w(\texttt{f}, \texttt{T}_i) <: ftype_w(\texttt{f}, \texttt{T}_j) \quad i \neq j \ \ i, j \in \{1,2\}}{ftype_w(\texttt{f}, \texttt{T}_1 \vee \texttt{T}_2) = ftype_w(\texttt{f}, \texttt{T}_i)}$$

$$\frac{\texttt{class C} \lhd \texttt{D}\ \{\ \overline{\texttt{T}\,\texttt{f}};\ \overline{\texttt{S}}\ \} \qquad \texttt{T t s}\{\ \texttt{e}\ \} \in \overline{\texttt{S}}}{stype(\texttt{s}, \texttt{C}) = \{\texttt{t}\}} \qquad \frac{\texttt{class C} \lhd \texttt{D}\ \{\ \overline{\texttt{T}\,\texttt{f}};\ \overline{\texttt{S}}\ \} \qquad \texttt{s} \notin \overline{\texttt{S}}}{stype(\texttt{s}, \texttt{C}) = stype(\texttt{s}, \texttt{D})}$$

$$stype(\texttt{s}, \texttt{T}_1 \vee \texttt{T}_2) = stype(\texttt{s}, \texttt{T}_1) \cup stype(\texttt{s}, \texttt{T}_2)$$

$$\frac{\texttt{class C} \lhd \texttt{D}\ \{\ \overline{\texttt{T}\,\texttt{f}};\ \overline{\texttt{S}}\ \} \qquad \texttt{T t s}\{\ \texttt{e}\ \} \in \overline{\texttt{S}}}{rtype(\texttt{s}, \texttt{C}) = \texttt{T}} \qquad \frac{\texttt{class C} \lhd \texttt{D}\ \{\ \overline{\texttt{T}\,\texttt{f}};\ \overline{\texttt{S}}\ \} \qquad \texttt{s} \notin \overline{\texttt{S}}}{rtype(\texttt{s}, \texttt{C}) = rtype(\texttt{s}, \texttt{D})}$$

$$rtype(\texttt{s}, \texttt{T}_1 \vee \texttt{T}_2) = rtype(\texttt{s}, \texttt{T}_1) \vee rtype(\texttt{s}, \texttt{T}_2)$$

$$\frac{\texttt{class C} \lhd \texttt{D}\ \{\ \overline{\texttt{T}\,\texttt{f}};\ \overline{\texttt{S}}\ \} \qquad \texttt{T t s}\{\ \texttt{e}\ \} \in \overline{\texttt{S}}}{sbody(\texttt{s}, \texttt{C}) = \texttt{e}} \qquad \frac{\texttt{class C} \lhd \texttt{D}\ \{\ \overline{\texttt{T}\,\texttt{f}};\ \overline{\texttt{S}}\ \} \qquad \texttt{s} \notin \overline{\texttt{S}}}{sbody(\texttt{s}, \texttt{C}) = sbody(\texttt{s}, \texttt{D})}$$

Fig. 5. Lookup Functions.

respectively, while *sbody* returns the body of a session. The *stype* function will return a singleton, in case it is invoked with a class name as argument. The interesting case is when it is invoked with a proper union type: it will return the union of the sets corresponding to the types of the classes that build the union type, so that we have all the session types (see how it is used in the type system, Figures 9 and 12). The *rtype* function behaves in a covariant way since the resulting object cannot be used in writing mode. Note that *sbody* is only invoked with a class name as type argument, since we invoke sessions on objects only, and an object has a class name as type.

It is easy to verify that all lookup functions applied to equivalent union types return either equivalent union types or the same sets of session types, whenever they are defined.

4. Operational Semantics

Objects passed in asynchronous communications are stored in a *heap*. A heap h is a finite mapping whose domain consists of objects and channel names. Its syntax is given by:

$$h \ ::= \ [] \ \mid \ \texttt{o} \mapsto (\texttt{C}, \overline{\texttt{f} = \texttt{o}}) \ \mid \ \texttt{k} \mapsto \overline{\texttt{o}} \ \mid \ h :: h$$

where :: denotes heap concatenation.

During evaluation, any expression $\texttt{new C}(\overline{\texttt{o}})$ will be replaced by a new object identifier o. The

$$\mathtt{e}\wr\mathtt{k}\wr = \begin{cases} \mathtt{e}_1\wr\mathtt{k}\wr;\mathtt{e}_2\wr\mathtt{k}\wr & \text{if } \mathtt{e} = \mathtt{e}_1;\mathtt{e}_2, \\ \mathtt{e}_1\wr\mathtt{k}\wr.\mathtt{f} & \text{if } \mathtt{e} = \mathtt{e}_1.\mathtt{f}, \\ \mathtt{e}_1\wr\mathtt{k}\wr.\mathtt{f}:=\mathtt{e}_2\wr\mathtt{k}\wr & \text{if } \mathtt{e} = \mathtt{e}_1.\mathtt{f}:=\mathtt{e}_2, \\ \mathtt{e}_1\wr\mathtt{k}\wr.\mathtt{s}\{\mathtt{e}_2\} & \text{if } \mathtt{e} = \mathtt{e}_1.\mathtt{s}\{\mathtt{e}_2\}, \\ \mathtt{e}_1\wr\mathtt{k}\wr\bullet\mathtt{s}\{\mathtt{k}\} & \text{if } \mathtt{e} = \mathtt{e}_1\bullet\mathtt{s}\{\ \}, \\ \mathtt{k}.\mathtt{sendC}(\mathtt{e}_0)\{\overline{\mathtt{C}\Rightarrow\mathtt{e}\wr\mathtt{k}\wr}\} & \text{if } \mathtt{e} = \mathtt{sendC}(\mathtt{e}_0)\{\overline{\mathtt{C}\Rightarrow\mathtt{e}}\}, \\ \mathtt{k}.\mathtt{recvC}(\mathtt{x})\{\overline{\mathtt{C}\Rightarrow\mathtt{e}\wr\mathtt{k}\wr}\} & \text{if } \mathtt{e} = \mathtt{recvC}(\mathtt{x})\{\overline{\mathtt{C}\Rightarrow\mathtt{e}}\}, \\ \mathtt{k}.\mathtt{sendW}(\mathtt{e}_0)\{\overline{\mathtt{C}\Rightarrow\mathtt{e}\wr\mathtt{k}\wr}\} & \text{if } \mathtt{e} = \mathtt{sendW}(\mathtt{e}_0)\{\overline{\mathtt{C}\Rightarrow\mathtt{e}}\}, \\ \mathtt{k}.\mathtt{recvW}(\mathtt{x})\{\overline{\mathtt{C}\Rightarrow\mathtt{e}\wr\mathtt{k}\wr}\} & \text{if } \mathtt{e} = \mathtt{recvW}(\mathtt{x})\{\overline{\mathtt{C}\Rightarrow\mathtt{e}}\}, \\ \mathtt{e} & \text{otherwise.} \end{cases}$$

Fig. 6. Channel Addition.

heap will then map the object identifier $\mathtt{o}$ to the pair $(\mathtt{C}, \overline{\mathtt{f}=\mathtt{o}})$ which consists of its class name $\mathtt{C}$ and the list of its fields with corresponding objects $\overline{\mathtt{o}}$; this mapping is denoted by $\mathtt{o}\mapsto(\mathtt{C},\overline{\mathtt{f}=\mathtt{o}})$.

The form $h[\mathtt{o}\mapsto h(\mathtt{o})[\mathtt{f}\mapsto\mathtt{o}']]$ denotes the update of the field $\mathtt{f}$ of the object $\mathtt{o}$ with the object $\mathtt{o}'$.

Channel names are mapped to queues of objects: $\mathtt{k}\mapsto\overline{\mathtt{o}}$. The heap produced by $h[\mathtt{k}\mapsto\overline{\mathtt{o}}]$ maps the channel $\mathtt{k}$ to the queue $\overline{\mathtt{o}}$. With some abuse of notation we write $\mathtt{o}::\overline{\mathtt{o}}$ and $\overline{\mathtt{o}}::\mathtt{o}$ to denote the queue whose first and last element is $\mathtt{o}$, respectively.

Heap membership for object identifiers and channels is checked using standard set notation, by identifying h with its domain, we can also write $\mathtt{o}\in h$, and $\mathtt{k}\in h$.

The queues of dual channels are used to exchange messages. A message receive on channel $\mathtt{k}$ takes the top object in the queue associated to $\mathtt{k}$, while a message send will add the object to the queue associated to $\tilde{\mathtt{k}}$. As usual $\tilde{\cdot}$ is an involution: $\tilde{\tilde{\mathtt{k}}}=\mathtt{k}$.

The values that can result from normal termination are parallel threads of objects.

In the reduction rules we make use of the special *channel addition* operation $\wr\ldots\wr$: its formal definition is in Figure 6, where $\{\overline{\mathtt{C}\Rightarrow\mathtt{e}}\}$ is short for $\{\mathtt{C}_1\Rightarrow\mathtt{e}_1\,[\!]\,\mathtt{C}_2\Rightarrow\mathtt{e}_2\}$. We denote by $\mathtt{e}\wr\mathtt{k}\wr$ the source expression $\mathtt{e}$ in which all occurrences of communication (receive, send) and delegation expressions which *are not* within the co-body of a session request are extended, so that they explicitly mention the channel $\mathtt{k}$ they will use (remember that channel names are not written by the programmer).

We use also $\mathtt{e}\lfloor\mathtt{e}'/\mathtt{cont}\rfloor$ to denote the expression $\mathtt{e}$ in which all expressions $\mathtt{cont}^{\mathtt{T}}$ which are free in $\mathtt{e}$, independently of the type annotations $\mathtt{T}$, are replaced by $\mathtt{e}'$. Thus this substitution preserves the correct nested structure of while expressions. We point out that the type annotation $\mathtt{T}$ of $\mathtt{cont}^{\mathtt{T}}$ plays no role in the evaluation, it is only used to guide the typechecker.

For example,

$$\mathtt{recvC}(\mathtt{x})\{\mathtt{C}_1\Rightarrow\mathtt{x}\,[\!]\,\mathtt{C}_2\Rightarrow\mathtt{cont}^{\mathtt{T}}\}\wr\mathtt{k}\wr\lfloor\mathtt{e}'/\mathtt{cont}\rfloor = \mathtt{k}.\mathtt{recvC}(\mathtt{x})\{\mathtt{C}_1\Rightarrow\mathtt{x}\,[\!]\,\mathtt{C}_2\Rightarrow\mathtt{e}'\}.$$

The reduction is a relation between pairs of threads and heaps:

$$P,h\longrightarrow P',h'$$

Reduction rules use evaluation contexts (based on run-time syntax) that capture the notion of

$$\textsc{Par-R}\ \frac{\mathtt{e},h \longrightarrow P,h'}{\mathtt{e} \mid\mid P_1,h \longrightarrow P \mid\mid P_1,h'} \qquad \textsc{Seq-R}\ \mathscr{E}[\mathtt{o};\mathtt{e}],h \longrightarrow \mathscr{E}[\mathtt{e}],h \qquad \textsc{Fld-R}\ \frac{h(\mathtt{o}) = (\mathtt{C},\overline{\mathtt{f}=\mathtt{o}})}{\mathscr{E}[\mathtt{o}.\mathtt{f}_i],h \longrightarrow \mathscr{E}[\mathtt{o}_i],h}$$

$$\textsc{NewC-R}\ \frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{T}\,\mathtt{f}} \qquad \mathtt{o} \notin h}{\mathscr{E}[\mathtt{new}\,\mathtt{C}(\overline{\mathtt{o}})],h \longrightarrow \mathscr{E}[\mathtt{o}],h::[\mathtt{o} \mapsto (\mathtt{C},\overline{\mathtt{f}=\mathtt{o}})]} \qquad \textsc{FldAss-R}\ \mathscr{E}[\mathtt{o}.\mathtt{f} := \mathtt{o}'],h \longrightarrow \mathscr{E}[\mathtt{o}'],h[\mathtt{o} \mapsto h(\mathtt{o})[\mathtt{f} \mapsto \mathtt{o}']]$$

$$\textsc{SessReq-R}\ \frac{h(\mathtt{o}) = (\mathtt{C},\_) \qquad \mathit{sbody}(\mathtt{s},\mathtt{C}) = \mathtt{e}' \qquad \mathtt{k},\tilde{\mathtt{k}} \notin h}{\mathscr{E}[\mathtt{o}.\mathtt{s}\,\{\mathtt{e}\}],h \longrightarrow \mathscr{E}[\mathtt{e}\{\mathtt{k}\}] \mid\mid [\mathtt{o}/\mathtt{this}]\mathtt{e}'\{\tilde{\mathtt{k}}\},h[\mathtt{k},\tilde{\mathtt{k}} \mapsto ()]} \qquad \textsc{SessDel-R}\ \frac{h(\mathtt{o}) = (\mathtt{C},\_) \qquad \mathit{sbody}(\mathtt{s},\mathtt{C}) = \mathtt{e}}{\mathscr{E}[\mathtt{o} \bullet \mathtt{s}\,\{\mathtt{k}\}],h \longrightarrow \mathscr{E}[[\mathtt{o}/\mathtt{this}]\mathtt{e}\{\mathtt{k}\}],h}$$

$$\textsc{SendCase-R}\ \frac{h(\tilde{\mathtt{k}}) = \overline{\mathtt{o}} \qquad h(\mathtt{o}) = (\mathtt{C},\_) \qquad \mathtt{C} \Downarrow \{\mathtt{C}_1,\mathtt{C}_2\} = \mathtt{C}_i}{\mathscr{E}[\mathtt{k}.\mathtt{sendC}(\mathtt{o})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}],h \longrightarrow \mathscr{E}[\mathtt{e}_i],h[\tilde{\mathtt{k}} \mapsto \overline{\mathtt{o}} :: \mathtt{o}]}$$

$$\textsc{ReceiveCase-R}\ \frac{h(\mathtt{k}) = \mathtt{o} :: \overline{\mathtt{o}} \qquad h(\mathtt{o}) = (\mathtt{C},\_) \qquad \mathtt{C} \Downarrow \{\mathtt{C}_1,\mathtt{C}_2\} = \mathtt{C}_i}{\mathscr{E}[\mathtt{k}.\mathtt{recvC}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}],h \longrightarrow \mathscr{E}[[\mathtt{o}/\mathtt{x}]\mathtt{e}_i],h[\mathtt{k} \mapsto \overline{\mathtt{o}}]}$$

$$\textsc{SendWhile-R}\ \mathscr{E}[\mathtt{k}.\mathtt{sendW}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}],h \longrightarrow \mathscr{E}[\mathtt{k}.\mathtt{sendC}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}'_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}'_2\}],h$$
$$\text{where } \mathtt{e}'_i = \mathtt{e}_i\lfloor \mathtt{k}.\mathtt{sendW}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}/\mathtt{cont}\rfloor$$

$$\textsc{ReceiveWhile-R}\ \mathscr{E}[\mathtt{k}.\mathtt{recvW}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}],h \longrightarrow \mathscr{E}[\mathtt{k}.\mathtt{recvC}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}'_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}'_2\}],h$$
$$\text{where } \mathtt{e}'_i = \mathtt{e}_i\lfloor \mathtt{k}.\mathtt{recvW}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}/\mathtt{cont}\rfloor$$

Fig. 7. Reduction Rules.

the "next subexpression to be reduced":

$$\begin{array}{rcl} \mathscr{E} & ::= & [-] \mid \mathscr{E};\mathtt{e} \mid \mathscr{E}.\mathtt{f} \mid \mathtt{new}\,\mathtt{C}(\overline{\mathtt{o}},\mathscr{E},\overline{\mathtt{e}}) \mid \mathscr{E}.\mathtt{f} := \mathtt{e} \mid \mathtt{o}.\mathtt{f} := \mathscr{E} \mid \mathscr{E}.\mathtt{s}\,\{\mathtt{e}\} \mid \\ & & \mathscr{E} \bullet \mathtt{s}\,\{\mathtt{k}\} \mid \mathtt{k}.\mathtt{sendC}(\mathscr{E})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} \end{array}$$

Reduction rules are presented in Figure 7, where any reducible expression is expressed as a composition of an evaluation context and a redex expression. The explicit mention of the evaluation context is needed in rule SessReq-R, in which a new thread is generated in parallel with the evaluation context. It is easy to verify that the set of redexes is defined by:

$$\begin{array}{c} \mathtt{o};\mathtt{e} \mid \mathtt{o}.\mathtt{f} \mid \mathtt{new}\,\mathtt{C}(\overline{\mathtt{o}}) \mid \mathtt{o}.\mathtt{f} := \mathtt{o} \mid \mathtt{o}.\mathtt{s}\,\{\mathtt{e}\} \mid \mathtt{o} \bullet \mathtt{s}\,\{\mathtt{k}\} \\ \mathtt{k}.\mathtt{sendC}(\mathtt{o})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} \mid \mathtt{k}.\mathtt{recvC}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} \\ \mathtt{k}.\mathtt{sendW}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} \mid \mathtt{k}.\mathtt{recvW}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} \end{array}$$

We call *delegation redexes* those of the shape $\mathtt{o} \bullet \mathtt{s}\,\{\mathtt{k}\}$ and *communication redexes* those of the last four shapes.

An arbitrary expression is equal to at most one evaluation context filled with one redex, and if it reduces, then there is exactly one reduction rule that applies. So the evaluation strategy is deterministic.

Rule Par-R models the execution of parallel threads. In this rule parallel composition is con-

sidered modulo structural equivalence. As usual, we define structural equivalence rules asserting that parallel composition is associative and commutative:

$$P \,||\, P_1 \equiv P_1 \,||\, P \quad P \,||\, (P_1 \,||\, P_2) \equiv (P \,||\, P_1) \,||\, P_2 \quad P \equiv P' \;\Rightarrow\; P \,||\, P_1 \equiv P' \,||\, P_1$$

Rule SESSREQ-R models the connection between the co-body $\mathtt{e}$ of a session request $\mathtt{o.s}\{\mathtt{e}\}$ and the body $\mathtt{e}'$ of the session $\mathtt{s}$, in the class of the object $\mathtt{o}$. This connection is established through a pair of fresh channels $\mathtt{k},\tilde{\mathtt{k}}$. For this purpose the expression $\mathtt{o.s}\{\mathtt{e}\}$ reduces, in the same context, to its own co-body $\mathtt{e}\wr\mathtt{k}\wr$ and in parallel, outside the context, it spawns the body $[\mathtt{o}/\mathtt{this}]\mathtt{e}'\wr\tilde{\mathtt{k}}\wr$ of the called session. The explicit substitution of $\mathtt{k}$ in $\mathtt{e}$ and of $\tilde{\mathtt{k}}$ in $\mathtt{e}'$ ensures that the communication uses the fresh dual channels $\mathtt{k}$ and $\tilde{\mathtt{k}}$. Thus, an object can serve *any number* of session requests. For example,

$$\begin{array}{c}\mathtt{o.s}\{\mathtt{sendC}(5)\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}\};\mathtt{new\ C}(\,) \longrightarrow \\ \mathtt{k.sendC}(5)\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1\wr\mathtt{k}\wr \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\wr\mathtt{k}\wr\};\mathtt{new\ C}(\,) \,||\, \\ \tilde{\mathtt{k}}.\mathtt{recvC}(\mathtt{x})\{\mathtt{C}'_1 \Rightarrow [\mathtt{o}/\mathtt{this}]\mathtt{e}'_1\wr\tilde{\mathtt{k}}\wr \,[]\, \mathtt{C}'_2 \Rightarrow [\mathtt{o}/\mathtt{this}]\mathtt{e}'_2\wr\tilde{\mathtt{k}}\wr\}\end{array}$$

if $\mathtt{recvC}(\mathtt{x})\{\mathtt{C}'_1 \Rightarrow \mathtt{e}'_1 \,[]\, \mathtt{C}'_2 \Rightarrow \mathtt{e}'_2\}$ is the body of session $\mathtt{s}$ in the class of the object $\mathtt{o}$.

Rule SESSDEL-R replaces the session delegation $\mathtt{o} \bullet \mathtt{s}\{\mathtt{k}\}$ by $[\mathtt{o}/\mathtt{this}]\mathtt{e}\wr\mathtt{k}\wr$, where $\mathtt{e}$ is the body of the session $\mathtt{s}$, in the class of the object $\mathtt{o}$. This allows the current session to be enriched by the capabilities provided by the session $\mathtt{s}$ of the object $\mathtt{o}$. The current thread executes the body $\mathtt{e}$ in which the current session channel $\mathtt{k}$ is used as subject for the communication, so that the delegation remains transparent for the thread using the dual channel $\tilde{\mathtt{k}}$. When the delegated job is over, the communication may continue within the current session, possibly using the value of $[\mathtt{o}/\mathtt{this}]\mathtt{e}\wr\mathtt{k}\wr$. Notice that, since the value produced by the execution of the delegated session body may be used after the delegation is over, we need both the return type and the session type of that body. And this is why we kept them both in the declaration of a session. See the explanation of Example 2.1 and the session declaration syntax in Figure 1. For instance

$$\mathtt{o} \bullet \mathtt{s}\{\mathtt{k}\} \longrightarrow \mathtt{k.recvC}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow [\mathtt{o}/\mathtt{this}]\mathtt{e}_1\wr\mathtt{k}\wr \,[]\, \mathtt{C}_2 \Rightarrow [\mathtt{o}/\mathtt{this}]\mathtt{e}_2\wr\mathtt{k}\wr\}$$

if $\mathtt{recvC}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}$ is the body of session $\mathtt{s}$ in the class of the object $\mathtt{o}$.

To sum up we can say that:

— *session invocation* creates a new channel and spawns the body of the called session;
— *session delegation* gives the active channel to another session whose body is executed in the same thread.

Since channels are implicit, only one session can be executed at a given time and the only possible interleaving of session is *nesting*. A session can be started while executing another session, but must complete before resuming the (outer scoped) previous session, hence we have nesting, but not general interleaving. This is the main reason why the progress property holds for communications in our calculus (see Theorem 6.2).

The communication rule for `sendC`, SENDCASE-R, puts the object $\mathtt{o}$ in the queue associated to the dual channel $\tilde{\mathtt{k}}$ of the communication channel $\mathtt{k}$. The computation then proceeds with the expression $\mathtt{e}_i$, if $\mathtt{C}_1 \neq \mathtt{C}_2$ and $\mathtt{C}_i$ is the smallest class in $\{\mathtt{C}_1,\mathtt{C}_2\}$ to which the object $\mathtt{o}$ belongs. Otherwise, if $\mathtt{C}_1 = \mathtt{C}_2$ and $\mathtt{o}$ belongs to $\mathtt{C}_1$, then the computation proceeds with $\mathtt{e}_1$. This is given by the condition $h(\mathtt{o}) = (\mathtt{C},\_)$ and by the following definition of $\mathtt{C} \Downarrow \{\mathtt{C}_1,\mathtt{C}_2\} = \mathtt{C}_i$, using the

subtyping relation (Figure 4):

$$\mathtt{C} \Downarrow \{\mathtt{C}_1,\mathtt{C}_2\} = \begin{cases} \mathtt{C}_i & \text{if } \ \mathtt{C} <: \mathtt{C}_i \ \text{ and } \ [\mathtt{C} <: \mathtt{C}_j \text{ with } i \neq j \text{ implies } \ \mathtt{C}_j \not<: \mathtt{C}_i], \\ \mathtt{C}_1 & \text{if } \ \mathtt{C} <: \mathtt{C}_1 = \mathtt{C}_2, \\ \bot & \text{otherwise.} \end{cases}$$

Notice that the only motivation for selecting the smallest index is to avoid introducing non-deterministic choices. In a more realistic context, for instance, we could adopt linguistic restrictions on the expressions $\mathtt{e}_i$, e.g., the condition $\mathtt{e}_1 = \mathtt{e}_2$ whenever $\mathtt{C}_1 = \mathtt{C}_2$. Dually the receive communication rule takes an object $\mathtt{o}$ from the queue associated to the channel $\mathtt{k}$ and returns the expression $[\mathtt{o}/\mathtt{x}]\mathtt{e}_i$, if $h(\mathtt{o}) = (\mathtt{C}, \_)$ and $\mathtt{C} \Downarrow \{\mathtt{C}_1,\mathtt{C}_2\} = \mathtt{C}_i$.

In rules SENDCASE-R and RECEIVECASE-R it is understood that the transition cannot fire if $\mathtt{C} \Downarrow \{\mathtt{C}_1,\mathtt{C}_2\} = \bot$. However we will see that $\mathtt{C} \Downarrow \{\mathtt{C}_1,\mathtt{C}_2\}$ is always defined in well-typed expressions.

Rules SENDWHILE-R and RECEIVEWHILE-R simply realise the repetition using the case communication expressions, in which the `sendW` and `recvW` expressions are unfolded in $\mathtt{e}_1$ and $\mathtt{e}_2$. Observe that $\mathtt{sendW}(\mathscr{E})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}$ is not an evaluation context, since we do not reduce the expression which controls the loop before the application of rule SENDWHILE-R. Then the application of rules SENDWHILE-R and RECEIVEWHILE-R cannot create any free occurrence of $\mathtt{cont}^{\mathtt{T}}$.

We will write $P,h \longrightarrow P',h'$ to mean that either P is a parallel composition and $P,h \longrightarrow P',h'$ is obtained by rule PAR-R (i.e. by reducing one of the expressions that are in parallel in P) or P is an expression $\mathtt{e}$ which reduces to P' by a reduction rule different from PAR-R.
As standard, the multi-step reduction $\longrightarrow^*$ is the reflexive, transitive closure of $\longrightarrow$.

We point out that communication and delegation expressions are reduced if and only if they contain explicit channels. So, for example, $\mathtt{sendC(o)}\{\ldots\}$ and $\mathtt{o} \bullet \mathtt{s}\{\}$ are stuck. We say that an expression $\mathtt{e}$ is *channel-complete* if all communication and delegation expressions of $\mathtt{e}$ without explicit channels occur inside session co-bodies. The shapes of closed and channel-complete expressions can be easily characterised by looking at the syntax of $\mathscr{F}\mathsf{SAM}^{\vee}$ (Figure 1).

Proposition 4.1. A closed and channel-complete expression is either an object identifier or an evaluation context filled with one redex.

By inspecting rules in Figure 7, it is easy to verify that no reduction can create new free variables or destroy the channel-completeness starting from the empty heap.

Proposition 4.2. If $\mathtt{e}$ is closed and channel-complete and $\mathtt{e}, [] \longrightarrow^* \mathtt{e}' \mid\mid P,h$, then $\mathtt{e}'$ is closed and channel-complete.

5. Typing

We consider two type systems, the first one for user expressions with occurrences of object identifiers, which are not directly expressible in the user syntax. We call these expressions *channel free expressions*. The second system types run-time expressions. The choice of considering channel free expressions instead of user expressions simplifies the formulation of the run-time typing rules, as we will see in Subsection 5.2.

$$\varepsilon \bowtie \varepsilon \qquad \alpha \bowtie \alpha \qquad \frac{\mathtt{t}_1 \bowtie \mathtt{t}'_1 \quad \mathtt{t}_2 \bowtie \mathtt{t}'_2}{\mathtt{t}_1.\mathtt{t}_2 \bowtie \mathtt{t}'_1.\mathtt{t}'_2}$$

$$\frac{\mathtt{C}_1 \vee \mathtt{C}_2 <: \mathtt{C}'_1 \vee \mathtt{C}'_2 \quad \mathtt{C}_i \Downarrow \{\mathtt{C}'_1, \mathtt{C}'_2\} = \mathtt{C}'_j \Rightarrow \mathtt{t}_i \bowtie \mathtt{t}'_j \quad \mathtt{C}'_l \Downarrow \{\mathtt{C}_1, \mathtt{C}_2\} = \mathtt{C}_k \Rightarrow \mathtt{t}_k \bowtie \mathtt{t}'_l}{\mu\alpha.!\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\} \bowtie \mu\alpha.?\{\mathtt{C}'_1 \Rightarrow \mathtt{t}'_1 \,[]\, \mathtt{C}'_2 \Rightarrow \mathtt{t}'_2\}}$$

Fig. 8. Duality Relation.

We say that a session type is *cont-free* if it does not contain occurrences of free session type variables and of ◎. Therefore, each cont-free session type has one of the following shapes:
— ε;
— $\mu\alpha.\dagger\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}$ or $\dagger\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}$;
or a concatenation of the session types above. For simplicity we will use in definitions unfolded recursive types whenever possible.

5.1. *Typing of Channel Free Expressions*

In this subsection we consider channel free expressions. The term environments therefore will contain also type assignments to object identifiers. The typing judgement has the shape

$$\Gamma \vdash \mathtt{e} : \mathtt{T} \fatsemi \mathtt{t}$$

where Γ is a term environment, which maps `this`, standard term variables and objects to types `T`, and `t` represents the session type of the (implicit) active channel. We observe that closed expressions can contain object identifiers, and therefore term environments having those object identifiers in their domain are required to type them (differently from the usual notion of closed expressions, which are typable from empty environments).

To guarantee a safe communication between two threads we must require their session types to be *dual*, i.e., that each send will correspond to a receive and vice-versa. The duality is then the symmetric relation generated by the rules of Figure 8, in which we consider folded recursive types, otherwise the definition would not be well founded. The exchanged values must also be of one of the classes expected by the receiver. All possible choices on the basis of the class of the exchanged value must continue with session types which are dual of each other. For this reason we have to perform checks on the type of the exchanged values in both directions:
— for any sent value of type $\mathtt{C}_i$ such that $\mathtt{C}_i \Downarrow \{\mathtt{C}'_1, \mathtt{C}'_2\} = \mathtt{C}'_j$ for some $1 \leq j \leq 2$ we require $\mathtt{t}_i \bowtie \mathtt{t}'_j$;
— for any received value of type $\mathtt{C}'_l$ such that $\mathtt{C}'_l \Downarrow \{\mathtt{C}_1, \mathtt{C}_2\} = \mathtt{C}_k$ for some $1 \leq k \leq 2$ we require $\mathtt{t}_k \bowtie \mathtt{t}'_l$.

For instance, let us consider the session types $!\{\mathtt{Shape} \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{String} \Rightarrow \mathtt{t}_2\}$ and $?\{\mathtt{Triangle} \Rightarrow \mathtt{t}_3 \,[]\, \mathtt{Object} \Rightarrow \mathtt{t}_4\}$ where `Triangle` $<:$ `Shape`. At run time a `Triangle` can be sent as a `Shape`, thus the types $\mathtt{t}_1$ and $\mathtt{t}_3$ have to be dual. Moreover, both a `Shape`, which is not a subclass of `Triangle`, and a `String` can be seen as `Object`s, thus both $\mathtt{t}_1$ and $\mathtt{t}_2$ must be dual of $\mathtt{t}_4$. Notice that, thanks to the absence of generics we can be more flexible than (Capecchi et al., 2009): the types used in the choices (actually their union) of a send can be subtypes of the ones expected (in the dual receive).

$$\textsc{Axiom-T}\quad \Gamma \vdash \mathtt{z} : \Gamma(\mathtt{z}) \fatsemi \varepsilon \qquad \textsc{Cont-T}\quad \Gamma \vdash \mathtt{cont}^{\mathtt{T}} : \mathtt{T} \fatsemi \circledcirc$$

$$\textsc{Sub-T}\quad \frac{\Gamma \vdash \mathtt{e} : \mathtt{T} \fatsemi \mathtt{t} \qquad \mathtt{T} <: \mathtt{T}'}{\Gamma \vdash \mathtt{e} : \mathtt{T}' \fatsemi \mathtt{t}}$$

$$\textsc{NewC-T}\quad \frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{T}\,\mathtt{f}} \qquad \Gamma \vdash \mathtt{e}_i : \mathtt{T}_i \fatsemi \varepsilon}{\Gamma \vdash \mathtt{new}\,\mathtt{C}(\overline{\mathtt{e}}) : \mathtt{C} \fatsemi \varepsilon}$$

$$\textsc{Fld-T}\quad \frac{\Gamma \vdash \mathtt{e} : \mathtt{T} \fatsemi \mathtt{t}}{\Gamma \vdash \mathtt{e.f} : \mathit{ftype}_r(\mathtt{f},\mathtt{T}) \fatsemi \mathtt{t}}$$

$$\textsc{Seq-T}\quad \frac{\Gamma \vdash \mathtt{e} : \mathtt{T} \fatsemi \mathtt{t} \qquad \Gamma \vdash \mathtt{e}' : \mathtt{T}' \fatsemi \mathtt{t}'}{\Gamma \vdash \mathtt{e;e}' : \mathtt{T}' \fatsemi \mathtt{t.t}'}$$

$$\textsc{FldAss-T}\quad \frac{\Gamma \vdash \mathtt{e} : \mathtt{T} \fatsemi \mathtt{t} \qquad \Gamma \vdash \mathtt{e}' : \mathit{ftype}_w(\mathtt{f},\mathtt{T}) \fatsemi \mathtt{t}'}{\Gamma \vdash \mathtt{e.f} := \mathtt{e}' : \mathit{ftype}_r(\mathtt{f},\mathtt{T}) \fatsemi \mathtt{t.t}'}$$

$$\textsc{SessReq-T}\quad \frac{\Gamma \vdash \mathtt{e} : \mathtt{T} \fatsemi \mathtt{t} \qquad \Gamma \vdash \mathtt{e}' : \mathtt{T}' \fatsemi \mathtt{t}' \qquad \mathtt{t}' \bowtie \mathtt{t}'' \ \forall \mathtt{t}'' \in \mathit{stype}(\mathtt{s},\mathtt{T})}{\Gamma \vdash \mathtt{e.s}\{\mathtt{e}'\} : \mathtt{T}' \fatsemi \mathtt{t}}$$

$$\textsc{SessDel-T}\quad \frac{\Gamma \vdash \mathtt{e} : \mathtt{T} \fatsemi \mathtt{t} \qquad \mathit{stype}(\mathtt{s},\mathtt{T}) = \{\mathtt{t}'\} \qquad \mathtt{t}' \neq \varepsilon \qquad \mathit{rtype}(\mathtt{s},\mathtt{T}) = \mathtt{T}'}{\Gamma \vdash \mathtt{e} \bullet \mathtt{s}\{\} : \mathtt{T}' \fatsemi \mathtt{t.t}'}$$

$$\textsc{SendC-T}\quad \frac{\Gamma \vdash \mathtt{e} : \mathtt{C}_1 \vee \mathtt{C}_2 \fatsemi \varepsilon \qquad \Gamma \vdash \mathtt{e}_i : \mathtt{T} \fatsemi \mathtt{t}_i}{\Gamma \vdash \mathtt{sendC}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T} \fatsemi !\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}}$$

$$\textsc{ReceiveC-T}\quad \frac{\Gamma, \mathtt{x} : \mathtt{C}_i \vdash \mathtt{e}_i : \mathtt{T} \fatsemi \mathtt{t}_i}{\Gamma \vdash \mathtt{recvC}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T} \fatsemi ?\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}}$$

$$\textsc{SendW-T}\quad \frac{\Gamma \vdash \mathtt{e} : \mathtt{C}_1 \vee \mathtt{C}_2 \fatsemi \varepsilon \qquad \Gamma \vdash \mathtt{e}_i : \mathtt{T} \fatsemi \mathtt{t}_i \qquad \mathtt{T} <: \mathtt{T}' \qquad \forall \mathtt{T}' \in \mathit{tc}(\mathtt{e}_1) \cup \mathit{tc}(\mathtt{e}_2) \qquad \alpha \text{ fresh in } \mathtt{t}_1, \mathtt{t}_2}{\Gamma \vdash \mathtt{sendW}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T} \fatsemi \mu\alpha.!\{\mathtt{C}_1 \Rightarrow [\alpha/\circledcirc]\mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow [\alpha/\circledcirc]\mathtt{t}_2\}}$$

$$\textsc{ReceiveW-T}\quad \frac{\Gamma, \mathtt{x} : \mathtt{C}_i \vdash \mathtt{e}_i : \mathtt{T} \fatsemi \mathtt{t}_i \qquad \mathtt{T} <: \mathtt{T}' \qquad \forall \mathtt{T}' \in \mathit{tc}(\mathtt{e}_1) \cup \mathit{tc}(\mathtt{e}_2) \qquad \alpha \text{ fresh in } \mathtt{t}_1, \mathtt{t}_2}{\Gamma \vdash \mathtt{recvW}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T} \fatsemi \mu\alpha.?\{\mathtt{C}_1 \Rightarrow [\alpha/\circledcirc]\mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow [\alpha/\circledcirc]\mathtt{t}_2\}}$$

Fig. 9. Typing Rules for Channel Free Expressions. The function *tc* is defined in Figure 10.

$$\mathit{tc}(\mathtt{e}) = \begin{cases} \mathit{tc}(\mathtt{e}_1) \cup \mathit{tc}(\mathtt{e}_2) & \text{if } \mathtt{e} = \mathtt{e}_1\mathtt{;}\mathtt{e}_2, \\ & \mathtt{e} = \mathtt{e}_1\mathtt{.f:=}\mathtt{e}_2, \\ & \mathtt{e} = \mathtt{e}_1\mathtt{.s}\{\mathtt{e}_2\}, \\ & \mathtt{e} = \mathtt{k.sendC}(\mathtt{e}_0)\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}, \\ & \mathtt{e} = \mathtt{k.recvC}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}, \\ \mathit{tc}(\mathtt{e}_1) & \text{if } \mathtt{e} = \mathtt{e}_1\mathtt{.f}, \\ & \mathtt{e} = \mathtt{e}_1 \bullet \mathtt{s}\{\mathtt{k}\}, \\ \{\mathtt{T}\} & \text{if } \mathtt{e} = \mathtt{cont}^{\mathtt{T}}, \\ \emptyset & \text{otherwise.} \end{cases}$$

Fig. 10. The function *tc*.

Typing rules for channel free expressions are in Figure 9.

By axiom Cont-T, $\mathtt{cont}^{\mathtt{T}}$ has type $\mathtt{T}$ from any Γ, since it is explicitly decorated by its type $\mathtt{T}$.

Following the standard notion of object instantiation, in rule NEWC-T we require that the initialisation of an object does not involve communications.

In rule SEQ-T we use session type concatenation to represent that first the communications in $\mathtt{e}$ and then those in $\mathtt{e}'$ are performed.

Rule FLDASS-T exploits the writing and reading uses of $\mathtt{e}'$.

The rule for session request SESSREQ-T relies on the duality relation (Figure 8) to ensure that all the bodies of the session $\mathtt{s}$ in the classes which build the type $\mathtt{T}$ and the co-body $\mathtt{e}'$ of the request will communicate properly. Since $\circledcirc$ has no dual session type, this rule ensures that there are no free occurrences of $\mathtt{cont}^{\mathtt{T}}$ in session bodies and co-bodies. For this reason, in well-typed expressions the reduction rules SENDWHILE-R and RECEIVEWHILE-R never replace $\mathtt{cont}^{\mathtt{T}}$ in session bodies and co-bodies.

In typing session delegation (rule SESSDEL-T) we take into account that the whole expression will be replaced by the session body defined in the class of the expression to which the session is delegated (cf. the reduction rule SESSDEL-R, Figure 7). Notice that the condition $\mathit{stype}(\mathtt{s},\mathtt{T})=\{\mathtt{t}'\}$ does not imply $\mathtt{T}$ be one class, but only that all definitions of $\mathtt{s}$ in the classes which build $\mathtt{T}$ have the same session types. Moreover, if a session has session type ε, then it is meaningless to use it in a delegation (while it is sensible to use it in a request). For this reason we require $\mathtt{t}' \neq \varepsilon$ in rule SESSDEL-T.

In the rules for communication expressions (SENDC-T, RECEIVEC-T, SENDW-T and RECEIVEW-T) the alternative branches $\mathtt{e}_i$ are both given type $\mathtt{T}$; however, this does not request both to have the same type, since $\mathtt{T}$ can be a proper union type. For instance, we may have that $\Gamma \vdash \mathtt{e}_1 : \mathtt{T}_1 \fatsemi \mathtt{t}_1$ and $\Gamma \vdash \mathtt{e}_2 : \mathtt{T}_2 \fatsemi \mathtt{t}_2$; by subsumption (rule SUB-T) we also have that $\Gamma \vdash \mathtt{e}_1 : \mathtt{T}_1 \vee \mathtt{T}_2 \fatsemi \mathtt{t}_1$ and $\Gamma \vdash \mathtt{e}_2 : \mathtt{T}_1 \vee \mathtt{T}_2 \fatsemi \mathtt{t}_2$. Then, $\mathtt{T} = \mathtt{T}_1 \vee \mathtt{T}_2$. Without union types the typing rules for these constructs in (Drossopoulou et al., 2007) were much more demanding and less clear.

Rules SENDW-T and RECEIVEW-T take into account that the free occurrences of $\mathtt{cont}^{\mathtt{T}}$ in the bodies are used to make recursive calls of the whole expression. This means that the type decorations of all these occurrences must be greater than or equal to the resulting type of the whole expression. This property is checked by the condition $\mathtt{T} <: \mathtt{T}'$ for all $\mathtt{T}' \in \mathit{tc}(\mathtt{e}_1) \cup \mathit{tc}(\mathtt{e}_2)$, using the function *tc* (defined in Figure 10). Moreover, the resulting session type is obtained by replacing the occurrences of $\circledcirc$ by a fresh variable α which is bound by the μ operator.

Observe that, in rules SENDC-T and SENDW-T, typing $\mathtt{e}$ with session type ε prevents $\mathtt{e}$ from containing occurrences of communications and $\mathtt{cont}^{\mathtt{T}}$. This restriction is not significant. If $\mathtt{e}$ contained communications, a possible dual for the `sendW` expression should be able to perform the dual communications at each iteration, before receiving the object that would select its continuation; such a dual should be of the form $\mathtt{e}'$;`receiveW(x)`$\{\mathtt{C}_1 \Rightarrow \ldots; \mathtt{e}'; \mathtt{cont}^{\mathtt{T}}; \ldots. \,[\!]\, \mathtt{C}_2 \Rightarrow \ldots;\}$, where $\mathtt{e}'$ contains the dual communications of $\mathtt{e}$. This suggests how a `sendW` expression with communications inside the argument can be encoded in our system. In order to maintain a sort of symmetry, we have the above restriction also in the typing of `sendC`. Let us notice that this problem concerns only communications in the current sessions but does not involve new sessions opened in $\mathtt{e}$: in fact, the typing allows $\mathtt{e}$ to contain session requests.

Figure 11 defines well-formed class tables. Rule SESS-WF says that a session declaration in a class $\mathtt{C}$ is well typed if its body has the declared return type and session type by assuming that `this` is of type $\mathtt{C}$. Notice that $\circledcirc$ has no dual type, so sessions whose bodies would be typed

$$\textsc{Sess-WF}\ \frac{\{\mathtt{this} : \mathtt{C}\} \vdash \mathtt{e} : \mathtt{T} \fatsemi \mathtt{t} \qquad \mathtt{t} \text{ is cont-free}}{\mathtt{T}\,\mathtt{t}\,\mathtt{s}\{\ \mathtt{e}\ \}\ \mathtt{ok\ in\ C}} \qquad \textsc{Class-WF}\ \frac{\mathtt{D\ ok} \qquad \overline{\mathtt{S}}\ \mathtt{ok\ in\ C}}{\mathtt{class\ C} \lhd \mathtt{D}\ \{\ \overline{\mathtt{T}\,\mathtt{f}};\ \overline{\mathtt{S}}\ \}\ \mathtt{ok}}$$

Fig. 11. Well-formed Class Tables.

with types containing $\circledcirc$ would be useless. This justifies the condition that $\mathtt{t}$ must be cont-free in rule SESS-WF, which implies that well-typed session bodies do not contain free occurrences of $\mathtt{cont}^{\mathtt{T}}$.

We conclude the subsection by observing that the system presented in Figure 9, given a typable expression $\mathtt{e}$ and the related class table, actually *infers* the session type of $\mathtt{e}$. As a matter of fact, that system is itself an inference algorithm of the session type of an expression, that expects session types of sessions to be declared in the class table.

It is easy to prove by induction on typing rules that:

Proposition 5.1. If $\Gamma \vdash \mathtt{e} : \mathtt{T} \fatsemi \mathtt{t}$ and $\Gamma \vdash \mathtt{e} : \mathtt{T}' \fatsemi \mathtt{t}'$, then $\mathtt{t} = \mathtt{t}'$.

The unicity of session types follows from the fact that receiving actions are modelled through expressions in which the classes of received objects are explicitly declared. This is a characterising feature of our approach to session types w.r.t. to standard systems (Yoshida and Vasconcelos, 2007).

In Section 7 we will present an inference algorithm that reconstructs the session types of session declarations, given a class table where these session types are omitted.

5.2. *Typing of Run-time Expressions*

During evaluation of well-typed programs, channel names are made explicit in send and receive expressions, as well as in session delegation expressions. Thus, in order to show how well-typedness is preserved under evaluation, we need to define new typing rules for run-time expressions. Furthermore, in typing run-time expressions, we must take into account the session types of more than one channel: run-time expressions contain explicit channel names (used for communications), thus session types must be associated with channel names in an appropriate way. Then judgements have the form

$$\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \Sigma$$

where Σ denotes a *session environment* which maps channels to session types.

A session environment maps only a finite set of channels to session types different from ε, and all the remaining to ε. We can then represent one session environment with an infinite number of finite sets which give all the meaningful associations and some of the others. For example $\{\mathtt{k} : \mathtt{t}\}$ and $\{\mathtt{k} : \mathtt{t}, \mathtt{k}' : \varepsilon\}$ represent the same environment. This choice avoids an explicit weakening rule for session environments.

Figure 12 gives the typing rules for run-time expressions, which differ from those for channel free expressions by having session environments instead of a unique session type. For this reason we extend the *concatenation* of session types to *session environments* as follows:

$$\Sigma.\Sigma'(\mathtt{k}) = \Sigma(\mathtt{k}).\Sigma'(\mathtt{k}).$$

$$\textsc{Axiom-RT}\ \ \Gamma \vdash_{\mathtt{r}} \mathtt{z} : \Gamma(\mathtt{z}) \fatsemi \emptyset \qquad \textsc{Cont-RT}\ \ \Gamma \vdash_{\mathtt{r}} \mathtt{cont}^{\mathtt{T}} : \mathtt{T} \fatsemi \{\mathtt{k} : \circledcirc\}$$

$$\textsc{Sub-RT}\ \ \frac{\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \Sigma \qquad \mathtt{T} <: \mathtt{T}'}{\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T}' \fatsemi \Sigma}$$

$$\textsc{NewC-RT}\ \ \frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{T}\,\mathtt{f}} \qquad \Gamma \vdash_{\mathtt{r}} \mathtt{e}_i : \mathtt{T}_i \fatsemi \Sigma_i}{\Gamma \vdash_{\mathtt{r}} \mathtt{new}\,\mathtt{C}(\overline{\mathtt{e}}) : \mathtt{C} \fatsemi \bigcup_i \Sigma_i}$$

$$\textsc{Fld-RT}\ \ \frac{\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \Sigma}{\Gamma \vdash_{\mathtt{r}} \mathtt{e.f} : \mathit{ftype}_r(\mathtt{f}, \mathtt{T}) \fatsemi \Sigma}$$

$$\textsc{Seq-RT}\ \ \frac{\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \Sigma \qquad \Gamma \vdash_{\mathtt{r}} \mathtt{e}' : \mathtt{T}' \fatsemi \Sigma'}{\Gamma \vdash_{\mathtt{r}} \mathtt{e};\mathtt{e}' : \mathtt{T}' \fatsemi \Sigma.\Sigma'}$$

$$\textsc{FldAss-RT}\ \ \frac{\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \Sigma \qquad \Gamma \vdash_{\mathtt{r}} \mathtt{e}' : \mathit{ftype}_w(\mathtt{f}, \mathtt{T}) \fatsemi \Sigma'}{\Gamma \vdash_{\mathtt{r}} \mathtt{e.f} := \mathtt{e}' : \mathit{ftype}_r(\mathtt{f}, \mathtt{T}) \fatsemi \Sigma.\Sigma'}$$

$$\textsc{SessReq-RT}\ \ \frac{\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \Sigma \qquad \Gamma \vdash \mathtt{e}' : \mathtt{T}' \fatsemi \mathtt{t}' \qquad \mathtt{t}' \bowtie \mathtt{t}''\ \forall \mathtt{t}'' \in \mathit{stype}(\mathtt{s}, \mathtt{T})}{\Gamma \vdash_{\mathtt{r}} \mathtt{e.s}\{\mathtt{e}'\} : \mathtt{T}' \fatsemi \Sigma}$$

$$\textsc{SessDel-RT}\ \ \frac{\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \Sigma \qquad \mathit{stype}(\mathtt{s}, \mathtt{T}) = \{\mathtt{t}\} \qquad \mathtt{t} \neq \varepsilon \qquad \mathit{rtype}(\mathtt{s}, \mathtt{T}) = \mathtt{T}'}{\Gamma \vdash_{\mathtt{r}} \mathtt{e} \bullet \mathtt{s}\{\mathtt{k}\} : \mathtt{T}' \fatsemi \Sigma.\{\mathtt{k} : \mathtt{t}\}}$$

$$\textsc{SendC-RT}\ \ \frac{\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{C}_1 \vee \mathtt{C}_2 \fatsemi \Sigma \qquad \Gamma \vdash_{\mathtt{r}} \mathtt{e}_i : \mathtt{T} \fatsemi \{\mathtt{k} : \mathtt{t}_i\}}{\Gamma \vdash_{\mathtt{r}} \mathtt{k.sendC}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T} \fatsemi \Sigma, \{\mathtt{k} : !\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}\}}$$

$$\textsc{ReceiveC-RT}\ \ \frac{\Gamma, \mathtt{x} : \mathtt{C}_i \vdash_{\mathtt{r}} \mathtt{e}_i : \mathtt{T} \fatsemi \{\mathtt{k} : \mathtt{t}_i\}}{\Gamma \vdash_{\mathtt{r}} \mathtt{k.recvC}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T} \fatsemi \{\mathtt{k} : ?\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}\}}$$

$$\textsc{SendW-RT}\ \ \frac{\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{C}_1 \vee \mathtt{C}_2 \fatsemi \emptyset \qquad \Gamma \vdash_{\mathtt{r}} \mathtt{e}_i : \mathtt{T} \fatsemi \{\mathtt{k} : \mathtt{t}_i\} \qquad \mathtt{T} <: \mathtt{T}' \qquad \forall \mathtt{T}' \in \mathit{tc}(\mathtt{e}_1) \cup \mathit{tc}(\mathtt{e}_2) \qquad \alpha \text{ fresh in } \mathtt{t}_1, \mathtt{t}_2}{\Gamma \vdash_{\mathtt{r}} \mathtt{k.sendW}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T} \fatsemi \{\mathtt{k} : \mu\alpha.!\{\mathtt{C}_1 \Rightarrow [\alpha/\circledcirc]\mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow [\alpha/\circledcirc]\mathtt{t}_2\}\}}$$

$$\textsc{ReceiveW-RT}\ \ \frac{\Gamma, \mathtt{x} : \mathtt{C}_i \vdash_{\mathtt{r}} \mathtt{e}_i : \mathtt{T} \fatsemi \{\mathtt{k} : \mathtt{t}_i\} \qquad \mathtt{T} <: \mathtt{T}' \qquad \forall \mathtt{T}' \in \mathit{tc}(\mathtt{e}_1) \cup \mathit{tc}(\mathtt{e}_2) \qquad \alpha \text{ fresh in } \mathtt{t}_1, \mathtt{t}_2}{\Gamma \vdash_{\mathtt{r}} \mathtt{k.recvW}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T} \fatsemi \{\mathtt{k} : \mu\alpha.?\{\mathtt{C}_1 \Rightarrow [\alpha/\circledcirc]\mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow [\alpha/\circledcirc]\mathtt{t}_2\}\}}$$

Fig. 12. Typing Rules for Run-time Expressions.

In rule NewC-RT the expressions for field initialisation can be partially evaluated, and for this reason they can contain channel names. For example

$$\mathtt{new}\,\mathtt{C}(\mathtt{o.s}\{\mathtt{sendC}(5)\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}\})$$

evaluates to

$$\mathtt{new}\,\mathtt{C}(\mathtt{k.sendC}(5)\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1\langle\mathtt{k}\rangle \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\langle\mathtt{k}\rangle\}) \mid\mid \tilde{\mathtt{k}}.\mathtt{recvC}(\mathtt{x})\{\mathtt{C}_1' \Rightarrow [\mathtt{o}/\mathtt{this}]\mathtt{e}_1'\langle\tilde{\mathtt{k}}\rangle \,[]\, \mathtt{C}_2' \Rightarrow [\mathtt{o}/\mathtt{this}]\mathtt{e}_2'\langle\tilde{\mathtt{k}}\rangle\}$$

if $\mathtt{recvC}(\mathtt{x})\{\mathtt{C}_1' \Rightarrow \mathtt{e}_1' \,[]\, \mathtt{C}_2' \Rightarrow \mathtt{e}_2'\}$ is the body of session $\mathtt{s}$ in the class of object $\mathtt{o}$.

In rule SessReq-RT we are making use of the judgement $\Gamma \vdash \mathtt{e}' : \mathtt{T} \fatsemi \mathtt{t}'$, where the expression $\mathtt{e}'$ does not contain channels, but it can contain object identifiers. For example by reducing

$$\mathtt{k.recvC}(\mathtt{x})\{\mathtt{C} \Rightarrow \mathtt{o.s}\{\mathtt{sendC}(\mathtt{x})\{\_\}\} \,[]\, \_\}, h$$

where $h(\mathtt{k}) = \mathtt{o}'$ and $h(\mathtt{o}') = (\mathtt{C}, \_)$, we get $\mathtt{o.s}\{\mathtt{sendC}(\mathtt{o}')\{\_\}\}$. This justifies our choice of con-

sidering channel free expressions instead of user expressions in the typing rules of the previous subsection.

In the typing of communications expressions, observe that the expressions $\mathtt{e}_1$ and $\mathtt{e}_2$, in the two branches, only contain the current channel $\mathtt{k}$ as subject, since channels are only created at run time and these expressions will never be reduced before the selection has been done. In rule SENDC-RT we know that the session environment Σ, obtained by typing the expression $\mathtt{e}$, cannot contain occurrences of the channel $\mathtt{k}$, since $\mathtt{e}$ is obtained by reducing a channel free expression with session type ε, as prescribed by rule SENDC-T. In rule SENDW-RT we can assume the emtpy session environment for typing the expression $\mathtt{e}$, since the evaluation of $\mathtt{e}$ cannot start before the `sendW` expression has been unfolded to a `sendC`.

The following lemma gives the weakening property for term environments.

Lemma 5.1 (Weakening). Let $\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \Sigma$.

1 If $\mathtt{x} \notin \mathsf{dom}(\Gamma)$ then $\Gamma, \mathtt{x} : \mathtt{T}' \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \Sigma$.
2 If $\mathtt{o} \notin \mathsf{dom}(\Gamma)$ then $\Gamma, \mathtt{o} : \mathtt{C} \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \Sigma$.

Proof. By induction on the derivation of $\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \Sigma$. □

The typing rules for run-time expressions differ from the ones for user expressions only in assigning the session type to explicit channels, not in the type $\mathtt{T}$.

The relation between the two systems is clarified by the following proposition that will be useful in showing the subject reduction property.

Proposition 5.2. $\Gamma \vdash \mathtt{e} : \mathtt{T} \fatsemi \mathtt{t}$ implies $\Gamma \vdash_{\mathtt{r}} \mathtt{e}\{\!|\mathtt{k}|\!\} : \mathtt{T} \fatsemi \{\mathtt{k} : \mathtt{t}\}$.

We notice that $\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \emptyset$ is equivalent to $\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \{\mathtt{k} : \varepsilon\}$ by our convention on session environments. Analogously, $\Sigma = \Sigma.\emptyset = \emptyset.\Sigma$ for any Σ.

As a final remark, we observe that we do not provide an explicit rule for typing parallel threads. Typing rules give type to single (run-time) expressions only, while expressions can reduce also to parallel threads by reduction rules. Indeed, in this case, we only use the notion of well-typedness: a parallel composition of expressions is considered to be well typed (in the environment Γ) if each single expression is typed (in Γ). We will take this point into account when formulating the subject reduction property in Section 6, where we prove that semantics preserves typing.

6. Properties

In this section we show the fundamental property ensuring that our system is well founded: type safety.

A program consists of a set of declarations and a main expression to be evaluated. Then a well-typed executable program means that the induced class table is well formed and the main expression is typed, using that class table, according to rules of Figure 9. We require the main expression to be a typable closed user expression; furthermore, all communication and delegation expressions must occur inside session co-bodies. It is easy to verify that this is equivalent to require typability in the system of Section 5.1 from the empty term environment with an empty session type, using a well-formed class table. Then we introduce the notion of initial expression as follows.

Definition 6.1. An *initial* expression $\mathtt{e}$ is such that $\emptyset \vdash \mathtt{e} : \mathtt{T} \fatsemi \varepsilon$ for some $\mathtt{T}$.

Proposition 6.1. An *initial* expression $\mathtt{e}$ is a closed and channel-complete user expression.

For example, the expressions $\mathtt{sendC(o)}\{\ldots\}$ and $\mathtt{o} \bullet \mathtt{s}\{\}$ are not initial expressions since if they are typed, then their term environments and their session types are not empty.

Proposition 5.2 guarantees that initial expressions are given the same type, with no assumption on communications, also by using the typing for run-time expressions.

The type safety property ensures that the evaluation of an initial expression cannot get stuck. We split the proof into two steps. First, we state the subject reduction property: namely, we prove that not only types are preserved but also the heap evolves in a consistent way with term and session environments, along the evaluation. Next, we prove type safety, dealing with the crucial case of communication expressions in order to show that they cannot get stuck on a communication deadlock.

6.1. *Subject Reduction*

For the proof of the Subject Reduction Theorem, we need some preliminary definitions and lemmas.

The first definition formalises the evolution of session types and session environments.

Definition 6.2.

1 A session type $\mathtt{t}'$ is at a *later stage* than another session type $\mathtt{t}$, $\mathtt{t} \sqsubseteq \mathtt{t}'$, if that is deducible from the following rules.

$$\textsc{Later-0}\ \frac{}{\mathtt{t} \sqsubseteq \varepsilon} \qquad \textsc{Later-1}\ \frac{}{\mathtt{t} \sqsubseteq \mathtt{t}} \qquad \textsc{Later-2}\ \frac{\mathtt{t} \sqsubseteq \mathtt{t}'' \quad \mathtt{t}'' \sqsubseteq \mathtt{t}'}{\mathtt{t} \sqsubseteq \mathtt{t}'} \qquad \textsc{Later-3}\ \frac{\mathtt{t} \sqsubseteq \mathtt{t}'}{\mathtt{t}.\mathtt{t}'' \sqsubseteq \mathtt{t}'.\mathtt{t}''} \qquad \textsc{Later-4}\ \frac{}{\dagger\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\} \sqsubseteq \mathtt{t}_i}$$

2 A session environment Σ' is at a *later stage* than another session environment Σ, $\Sigma \sqsubseteq \Sigma'$, if $\mathtt{k} : \mathtt{t} \in \Sigma$ and $\mathtt{t} \neq \varepsilon$ imply $\mathtt{k} : \mathtt{t}' \in \Sigma'$ and $\mathtt{t} \sqsubseteq \mathtt{t}'$.

Evolution of session environments takes also into account that new channels can be created by session calls, so for example assuming that $\mathtt{t}_3$ and $\mathtt{t}_4$ are dual:

$$\{\mathtt{k} : !\{\mathtt{Shape} \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{String} \Rightarrow \mathtt{t}_2\}\} \sqsubseteq \{\mathtt{k} : \mathtt{t}_1, \mathtt{k}_1 : \mathtt{t}_3, \tilde{\mathtt{k}}_1 : \mathtt{t}_4\}.$$

The second definition gives standard conditions on heap well-formedness and agreement between heaps and term environments.

Definition 6.3 (Well-Formed Heap and Agreement). A term environment Γ and a heap h *agree*, written $ag(\Gamma; h)$, if:

1 h is well formed:
$h(\mathtt{o}) = (\mathtt{C}, \overline{\mathtt{f} = \mathtt{o}})$, $\mathit{ftype}_r(\mathtt{C}, \mathtt{f}_i) = \mathtt{T} \Rightarrow h(\mathtt{o})(\mathtt{f}_i) = (\mathtt{C}', \_)$, $\mathtt{C}' <: \mathtt{T}$ and

2 the classes of objects in h are the classes associated to them by Γ:
$\forall \mathtt{o} \in \mathsf{dom}(\Gamma)$, $h(\mathtt{o}) = (\Gamma(\mathtt{o}), \_)$.

In point 1 of the above definition, recall that $\mathit{ftype}_r(\mathtt{C}, \mathtt{f}_i)$=$\mathit{ftype}_w(\mathtt{C}, \mathtt{f}_i)$, being $\mathtt{C}$ a class.

The following lemma states the obvious property that, in any type derivation ending by rule SUB-RT, there is a subderivation giving a subtype to the same expression such that its final rule is different from SUB-RT.

Lemma 6.1. In any derivation of $\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T}' \fatsemi \Sigma$ there is a subderivation of $\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \Sigma$, with $\mathtt{T} <: \mathtt{T}'$, where the last applied rule is different from SUB-RT.

Proof. By straightforward induction on the derivation of $\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \Sigma$. □

Hence, by Lemma 6.1, in the following proofs we will assume without loss of generality that the given typing derivations end with a rule different from SUB-RT.

In order to simplify the proof of Subject Reduction, it is handy to preliminarily show preservation of typing under substitution of subexpressions. In our calculus, the difficulty is that we must deal carefully with session environments in substitutions.

Lemma 6.2 uses evaluation contexts as defined in Section 4. Note that this lemma does not require that the expression in the hole of the context be a redex.

Lemma 6.2 (Evaluation Context Substitution). In any derivation of $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{e}] : \mathtt{T} \fatsemi \Sigma$, there exist Σ_1, Σ_2, $\mathtt{T}'$, such that

1 there is a subderivation of $\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T}' \fatsemi \Sigma_1$ and $\Sigma = \Sigma_1.\Sigma_2$,
2 $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{e}'] : \mathtt{T} \fatsemi \Sigma_1'.\Sigma_2$, for any $\mathtt{e}'$ such that $\Gamma \vdash_{\mathtt{r}} \mathtt{e}' : \mathtt{T}' \fatsemi \Sigma_1'$ with $\Sigma_1 \sqsubseteq \Sigma_1'$.

Proof. By induction on the definition of $\mathscr{E}$. The base case, that is when $\mathscr{E}$ is the empty context, is trivial. In the induction step, each case proceeds by analysing the final rule, used in the derivation of $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{e}] : \mathtt{T} \fatsemi \Sigma$, which is assumed to be different from SUB-RT by Lemma 6.1.

— $\mathscr{E}[\mathtt{e}] = \mathscr{E}'[\mathtt{e}]; \mathtt{e}''$. Immediate from rule SEQ-RTand the induction hypothesis.
— $\mathscr{E}[\mathtt{e}] = \mathscr{E}'[\mathtt{e}].\mathtt{s}\{\mathtt{e}''\}$. The final rule SESSREQ-RT implies that $\Sigma = \Sigma'.\Sigma''$ and there is a subderivation of $\Gamma \vdash_{\mathtt{r}} \mathscr{E}'[\mathtt{e}] : \mathtt{T}_1 \fatsemi \Sigma'$. Then, by the induction hypothesis, we have $\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T}' \fatsemi \Sigma_1$ where $\Sigma' = \Sigma_1.\Sigma_2'$, i.e. $\Sigma = \Sigma_1.\Sigma_2$ by taking $\Sigma_2 = \Sigma_2'.\Sigma''$. Moreover, by the induction hypothesis, $\Gamma \vdash_{\mathtt{r}} \mathscr{E}'[\mathtt{e}] : \mathtt{T}_1 \fatsemi \Sigma'$ implies $\Gamma \vdash_{\mathtt{r}} \mathscr{E}'[\mathtt{e}'] : \mathtt{T}_1 \fatsemi \Sigma_1'.\Sigma_2'$. Hence, we can substitute a derivation of $\Gamma \vdash_{\mathtt{r}} \mathscr{E}'[\mathtt{e}'] : \mathtt{T}_1 \fatsemi \Sigma_1'.\Sigma_2'$ for the subderivation of $\Gamma \vdash_{\mathtt{r}} \mathscr{E}'[\mathtt{e}] : \mathtt{T}_1 \fatsemi \Sigma_1.\Sigma_2'$, into the derivation of $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{e}] : \mathtt{T} \fatsemi \Sigma_1.\Sigma_2$. Rule SESSREQ-RT still applies since the other premises stay the same. So we obtain $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{e}'] : \mathtt{T} \fatsemi \Sigma_1'.\Sigma_2$.
— $\mathscr{E}[\mathtt{e}] = \mathscr{E}'[\mathtt{e}] \bullet \mathtt{s}\{\mathtt{k}\}$. The final rule SESSDEL-RT implies that there is a subderivation of $\Gamma \vdash_{\mathtt{r}} \mathscr{E}'[\mathtt{e}] : \mathtt{T}_1 \fatsemi \Sigma'$, such that $\Sigma = \Sigma'.\{\mathtt{k} : \mathtt{t}\}$, $\mathit{stype}(\mathtt{s}, \mathtt{T}_1) = \{\mathtt{t}\}$ and $\mathit{rtype}(\mathtt{s}, \mathtt{T}_1) = \mathtt{T}$. Then, by the induction hypothesis, we have a subderivation of $\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T}' \fatsemi \Sigma_1$ where $\Sigma' = \Sigma_1.\Sigma_2'$ for some Σ_2'; thus $\Sigma = \Sigma_1.\Sigma_2$ by defining $\Sigma_2 = \Sigma_2'.\{\mathtt{k} : \mathtt{t}\}$. Moreover, by the induction hypothesis, $\Gamma \vdash_{\mathtt{r}} \mathscr{E}'[\mathtt{e}] : \mathtt{T}_1 \fatsemi \Sigma'$ implies $\Gamma \vdash_{\mathtt{r}} \mathscr{E}'[\mathtt{e}'] : \mathtt{T}_1 \fatsemi \Sigma_1'.\Sigma_2'$. Hence we can replace $\Gamma \vdash_{\mathtt{r}} \mathscr{E}'[\mathtt{e}] : \mathtt{T}_1 \fatsemi \Sigma_1.\Sigma_2'$ with $\Gamma \vdash_{\mathtt{r}} \mathscr{E}'[\mathtt{e}'] : \mathtt{T}_1 \fatsemi \Sigma_1'.\Sigma_2'$ into the derivation of $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{e}] : \mathtt{T} \fatsemi \Sigma_1.\Sigma_2$ and rule SESSDEL-RT still applies. So we obtain $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{e}'] : \mathtt{T} \fatsemi \Sigma_1'.\Sigma_2$.
— $\mathscr{E}[\mathtt{e}] = \mathtt{k}.\mathtt{sendC}(\mathscr{E}'[\mathtt{e}])\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}$. The final rule is SENDC-RT which implies that there is a subderivation of $\Gamma \vdash_{\mathtt{r}} \mathscr{E}'[\mathtt{e}] : \mathtt{C}_1 \vee \mathtt{C}_2 \fatsemi \Sigma'$, such that $\Sigma = \Sigma', \{\mathtt{k} :! \{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}\}$ and $\Gamma \vdash_{\mathtt{r}} \mathtt{e}_i : \mathtt{T} \fatsemi \{\mathtt{k} : \mathtt{t}_i\}$. Then, by the induction hypothesis, there is a subderivation of $\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T}' \fatsemi \Sigma_1$ where $\Sigma' = \Sigma_1.\Sigma''$, for some Σ'', that is $\Sigma = \Sigma_1.\Sigma_2$ by taking $\Sigma_2 = \Sigma'', \{\mathtt{k} :! \{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}\}$. Moreover, the induction hypothesis on $\Gamma \vdash_{\mathtt{r}} \mathscr{E}'[\mathtt{e}] : \mathtt{C}_1 \vee \mathtt{C}_2 \fatsemi \Sigma_1.\Sigma''$ tells us that $\Gamma \vdash_{\mathtt{r}} \mathscr{E}'[\mathtt{e}'] : \mathtt{C}_1 \vee \mathtt{C}_2 \fatsemi \Sigma_1'.\Sigma''$. Hence, since

$\Gamma \vdash_{\mathtt{r}} \mathtt{e}_i : \mathtt{T} \fatsemi \{\mathtt{k} : \mathtt{t}_i\}$, we obtain $\Gamma \vdash_{\mathtt{r}} \mathtt{k}.\mathtt{sendC}(\mathscr{E}'[\mathtt{e}'])\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{C}_1 \vee \mathtt{C}_2 \fatsemi \Sigma'_1.\Sigma_2$, by rule SENDC-RT.

The remaining cases are straightforward, following the proof pattern of the cases above. □

Lemma 6.3 (Term Substitution).

1 If $\Gamma, \mathtt{z} : \mathtt{C} \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \Sigma$ and $\Gamma \vdash_{\mathtt{r}} \mathtt{o} : \mathtt{C} \fatsemi \emptyset$, then $\Gamma \vdash_{\mathtt{r}} [\mathtt{o}/\mathtt{z}]\mathtt{e} : \mathtt{T} \fatsemi \Sigma$.

2 If $\Gamma \vdash_{\mathtt{r}} \mathtt{k}.\mathtt{sendW}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T} \fatsemi \{\mathtt{k} : \mu\alpha.!\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}\}$, then, for $i \in \{1,2\}$,

$$\Gamma \vdash_{\mathtt{r}} \mathtt{e}_i\lfloor \mathtt{k}.\mathtt{sendW}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}/\mathtt{cont}\rfloor : \mathtt{T} \fatsemi \{\mathtt{k} : \mathtt{t}'_i\}$$

where $\mathtt{t}'_i = [\mu\alpha.!\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}/\alpha]\mathtt{t}_i$.

3 If $\Gamma, \mathtt{x} : \mathtt{C}_i \vdash_{\mathtt{r}} \mathtt{k}.\mathtt{recvW}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T} \fatsemi \{\mathtt{k} : \mu\alpha.?\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}\}$, then, for $i \in \{1,2\}$,

$$\Gamma, \mathtt{x} : \mathtt{C}_i \vdash_{\mathtt{r}} \mathtt{e}_i\lfloor \mathtt{k}.\mathtt{recvW}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}/\mathtt{cont}\rfloor : \mathtt{T} \fatsemi \{\mathtt{k} : \mathtt{t}'_i\}$$

where $\mathtt{t}'_i = [\mu\alpha.?\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}/\alpha]\mathtt{t}_i$.

Proof.

1 Immediate by substituting $\Gamma \vdash_{\mathtt{r}} \mathtt{o} : \mathtt{C} \fatsemi \emptyset$ for $\Gamma, \mathtt{z} : \mathtt{C} \vdash_{\mathtt{r}} \mathtt{z} : \mathtt{C} \fatsemi \emptyset$, in any derivation of $\Gamma, \mathtt{z} : \mathtt{C} \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \Sigma$.

2 By rule SENDW-RT

$$\Gamma \vdash_{\mathtt{r}} \mathtt{k}.\mathtt{sendW}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T} \fatsemi \{\mathtt{k} : \mu\alpha.!\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}\}$$

implies $\Gamma, \mathtt{x} : \mathtt{C}_i \vdash_{\mathtt{r}} \mathtt{e}_i : \mathtt{T}_i \fatsemi \{\mathtt{k} : \mathtt{t}''_i\}$, for $\mathtt{t}''_i = [\circledcirc/\alpha]\mathtt{t}_i$, and $\mathtt{T} <: \mathtt{T}'$ for all $\mathtt{T}' \in tc(\mathtt{e}_1) \cup tc(e_2)$. This last condition and the definition of *tc* ensure that if $\mathtt{cont}^{\mathtt{T}'}$ occurs free in $\mathtt{e}_1$ or $\mathtt{e}_2$, then $\mathtt{T} <: \mathtt{T}'$. Therefore any free occurrence of $\mathtt{cont}^{\mathtt{T}'}$ in $\mathtt{e}_i$ is given type by $\Gamma \vdash_{\mathtt{r}} \mathtt{cont}^{\mathtt{T}'} : \mathtt{T}' \fatsemi \{\mathtt{k} : \circledcirc\}$.
From $\Gamma \vdash_{\mathtt{r}} \mathtt{k}.\mathtt{sendW}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T} \fatsemi \{\mathtt{k} : \mu\alpha.!\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}\}$ we derive $\Gamma \vdash_{\mathtt{r}} \mathtt{k}.\mathtt{sendW}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T}' \fatsemi \{\mathtt{k} : \mu\alpha.!\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}\}$ by rule SUB-RT. Observe that the only constraint satisfied by $\circledcirc$ and which appears in the premises of typing rules is $\circledcirc \neq \varepsilon$, since no session type is dual of $\circledcirc$. Thus, if we replace $\Gamma \vdash_{\mathtt{r}} \mathtt{cont}^{\mathtt{T}'} : \mathtt{T}' \fatsemi \{\mathtt{k} : \circledcirc\}$ by $\Gamma \vdash_{\mathtt{r}} \mathtt{k}.\mathtt{sendW}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T}' \fatsemi \{\mathtt{k} : \mu\alpha.!\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}\}$, inside the derivation of $\Gamma, \mathtt{x} : \mathtt{C}_i \vdash_{\mathtt{r}} \mathtt{e}_i : \mathtt{T}_i \fatsemi \{\mathtt{k} : \mathtt{t}''_i\}$, we obtain a derivation of $\Gamma, \mathtt{x} : \mathtt{C}_i \vdash_{\mathtt{r}} \mathtt{e}_i\lfloor \mathtt{k}.\mathtt{sendW}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}/\mathtt{cont}\rfloor : \mathtt{T} \fatsemi \{\mathtt{k} : \mathtt{t}'_i\}$.

3 Similar to the previous point, using RECEIVEW-RT in place of SENDW-RT.

□

Lemma 6.4 (Typing of Session Bodies). If $sbody(\mathtt{s}, \mathtt{C}) = \mathtt{e}$, $stype(\mathtt{s}, \mathtt{C}) = \mathtt{t}$ and $rtype(\mathtt{s}, \mathtt{C}) = \mathtt{T}$, then $\{\mathtt{this} : \mathtt{C}\} \vdash \mathtt{e} : \mathtt{T} \fatsemi \mathtt{t}$.

Proof. By induction on the definition of $sbody(\mathtt{s}, \mathtt{C})$, using the definitions of *stype* and *rtype*. In the base case, $\mathtt{s}$ is defined in $\mathtt{C}$ and then the proof follows from rule SESS-WF. The induction step is straightforward. □

We can now prove the Subject Reduction Theorem. We type only single expressions, but they can result in parallel threads. Since we do not have a typing for parallel threads we require each single expression to be well typed. Moreover we want to get our property in the most general form, allowing the property to hold for all well-typed expressions, which sometimes can be generated by initial expressions only in parallel with other expressions. For example, no initial expression can reduce to the expression

$$\mathtt{e} = \mathtt{o.s}\{\mathtt{sendC(5)}\{\mathtt{e}_1\}\};\mathtt{k.sendC(3)}\{\mathtt{e}_2\langle\mathtt{k}\rangle\},$$

but

$$\mathtt{e}_0 = \mathtt{o}'.\mathtt{s}'\{\mathtt{o.s}\{\mathtt{sendC(5)}\{\mathtt{e}_1\}\};\mathtt{sendC(3)}\{\mathtt{e}_2\}\}$$

reduces to $\mathtt{e} \mid\mid \tilde{\mathtt{k}}.\mathtt{recvC}(\mathtt{x})\{[\mathtt{o}'/\mathtt{this}]\mathtt{e}'\langle\tilde{\mathtt{k}}\rangle\}$ if $\mathtt{recvC}(\mathtt{x})\{\mathtt{e}'\}$ is the body of session $\mathtt{s}'$ in the class of the object $\mathtt{o}'$.

Notice also that receive expressions can never get objects of wrong types. For example the execution of $\mathtt{k.recvC}(\mathtt{x})\{\mathtt{Bool} \Rightarrow \neg\mathtt{x} \,[]\, \mathtt{Int} \Rightarrow -\mathtt{x}\}$ if $h(\mathtt{k})=\mathtt{"a"}$ is simply stopped, i.e. it does not produce a run-time error. In fact the reduction rule RECEIVECASE-R requires the class of the object in the heap to be a subclass of at least one of the classes declared in the `recvC` expression. Note that such a configuration cannot be generated starting from an initial expression. For this reason, in contrast to the calculus of (Coppo et al., 2007), we do not need to require agreement between the objects in the queues associated to channels by the heap and the session types of the same channels in the session environment.

Theorem 6.1 (Subject Reduction). If $ag(\Gamma;h)$ and $\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \Sigma$ then

1 $\mathtt{e},h \longrightarrow \mathtt{e}',h'$ implies that there exist Σ', Γ' such that $\Gamma \subseteq \Gamma'$ and $\Sigma \sqsubseteq \Sigma'$, and $ag(\Gamma';h')$, and $\Gamma' \vdash_{\mathtt{r}} \mathtt{e}' : \mathtt{T} \fatsemi \Sigma'$.

2 $\mathtt{e},h \longrightarrow \mathtt{e}_1 \mid\mid \mathtt{e}_2,h'$ implies that $h' = h[\mathtt{k},\tilde{\mathtt{k}} \mapsto ()]$ for some fresh $\mathtt{k}$, and $ag(\Gamma;h')$, and that there exist $\mathtt{T}',\mathtt{t},\mathtt{t}'$ such that $\Gamma \vdash_{\mathtt{r}} \mathtt{e}_1 : \mathtt{T} \fatsemi \Sigma \cup \{\mathtt{k} : \mathtt{t}\}$, and $\Gamma \vdash_{\mathtt{r}} \mathtt{e}_2 : \mathtt{T}' \fatsemi \{\tilde{\mathtt{k}} : \mathtt{t}'\}$, and $\mathtt{t} \bowtie \mathtt{t}'$.

Proof. By induction on the definition of $\longrightarrow$. We proceed by case analysis.

By Lemma 6.1 we consider typing derivations of $\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{T} \fatsemi \Sigma$ where the last applied rule is different from SUB-RT.

Case SESSREQ-R.

$$\frac{h(\mathtt{o}) = (\mathtt{C},\_) \quad \mathit{sbody}(\mathtt{s},\mathtt{C}) = \mathtt{e}' \quad \mathtt{k},\tilde{\mathtt{k}} \notin h}{\mathscr{E}[\mathtt{o.s}\{\mathtt{e}\}],h \longrightarrow \mathscr{E}[\mathtt{e}\langle\mathtt{k}\rangle] \mid\mid [\mathtt{o}/\mathtt{this}]\mathtt{e}'\langle\tilde{\mathtt{k}}\rangle,h[\mathtt{k},\tilde{\mathtt{k}} \mapsto ()]}$$

By $h(\mathtt{o}) = (\mathtt{C},\_)$ and $ag(\Gamma;h)$ we get that $\Gamma \vdash_{\mathtt{r}} \mathtt{o} : \mathtt{C} \fatsemi \emptyset$ using AXIOM-RT.
Since, by hypothesis $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{o.s}\{\mathtt{e}\}] : \mathtt{T} \fatsemi \Sigma$, by Lemma 6.2(1), we have that
$\Gamma \vdash_{\mathtt{r}} \mathtt{o.s}\{\mathtt{e}\} : \mathtt{T}' \fatsemi \Sigma_1$ and $\Sigma = \Sigma_1.\Sigma_2$. From rule SESSREQ-RT we have that $\Sigma_1 = \emptyset$, $\Sigma_2 = \Sigma$, $\Gamma \vdash \mathtt{e} : \mathtt{T}' \fatsemi \mathtt{t}'$, $\mathit{stype}(\mathtt{s},\mathtt{C}) = \mathtt{t}$ and $\mathtt{t} \bowtie \mathtt{t}'$.
By Proposition 5.2 we get $\Gamma \vdash_{\mathtt{r}} \mathtt{e}\langle\mathtt{k}\rangle : \mathtt{T}' \fatsemi \{\mathtt{k} : \mathtt{t}'\}$.
By Lemma 6.2(2), we have that $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{e}\langle\mathtt{k}\rangle] : \mathtt{T} \fatsemi \{\mathtt{k} : \mathtt{t}'\}.\Sigma$.
Let $\mathit{rtype}(\mathtt{s},\mathtt{C}) = \mathtt{T}_0$, then $\mathtt{this} : \mathtt{C} \vdash \mathtt{e}' : \mathtt{T}_0 \fatsemi \mathtt{t}$ by Lemma 6.4, which implies
$\mathtt{this} : \mathtt{C} \vdash_{\mathtt{r}} \mathtt{e}'\langle\tilde{\mathtt{k}}\rangle : \mathtt{T}_0 \fatsemi \{\tilde{\mathtt{k}} : \mathtt{t}\}$, by Proposition 5.2. Therefore, by Lemmas 5.1 and 6.3(1), we

conclude $\Gamma \vdash_{\mathtt{r}} [\mathtt{o}/\mathtt{this}]\mathtt{e}'\{\!|\tilde{\mathtt{k}}|\!\} : \mathtt{T}_0 \S \{\tilde{\mathtt{k}} : \mathtt{t}\}$.

Notice that the new heap $h[\mathtt{k}, \tilde{\mathtt{k}} \mapsto ()]$ still agrees with Γ since the only changes are about channels.

Case SESSDEL-R.

$$\frac{h(\mathtt{o}) = (\mathtt{C}, \_) \quad \mathit{sbody}(\mathtt{s}, \mathtt{C}) = \mathtt{e}}{\mathscr{E}[\mathtt{o} \bullet \mathtt{s}\,\{\mathtt{k}\}], h \longrightarrow \mathscr{E}[[\mathtt{o}/\mathtt{this}]\mathtt{e}\{\!|\mathtt{k}|\!\}], h}$$

By $h(\mathtt{o}) = (\mathtt{C}, \_)$ and $ag(\Gamma; h)$ we get that $\Gamma \vdash_{\mathtt{r}} \mathtt{o} : \mathtt{C} \S \emptyset$ using AXIOM-RT.

Since, by hypothesis $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{o} \bullet \mathtt{s}\,\{\mathtt{k}\}] : \mathtt{T} \S \Sigma$, by Lemma 6.2(1), we have that $\Gamma \vdash_{\mathtt{r}} \mathtt{o} \bullet \mathtt{s}\,\{\mathtt{k}\} : \mathtt{T}' \S \Sigma_1$ and $\Sigma = \Sigma_1.\Sigma_2$. From rule SESSDEL-RT we have that $\Sigma_1 = \{\mathtt{k} : \mathtt{t}\}$, $\mathit{stype}(\mathtt{s}, \mathtt{C}) = \mathtt{t}$ and $\mathit{rtype}(\mathtt{s}, \mathtt{C}) = \mathtt{T}'$.

By Lemma 6.4, $\mathtt{this} : \mathtt{C} \vdash \mathtt{e} : \mathtt{T}' \S \mathtt{t}$, and then by Proposition 5.2 $\mathtt{this} : \mathtt{C} \vdash_{\mathtt{r}} \mathtt{e}\{\!|\mathtt{k}|\!\} : \mathtt{T}' \S \{\mathtt{k} : \mathtt{t}\}$. Therefore, by Lemmas 5.1 and 6.3(1), we have that $\Gamma \vdash_{\mathtt{r}} [\mathtt{o}/\mathtt{this}]\mathtt{e}\{\!|\mathtt{k}|\!\} : \mathtt{T}' \S \{\mathtt{k} : \mathtt{t}\}$. By Lemma 6.2(2), we conclude $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[[\mathtt{o}/\mathtt{this}]\mathtt{e}\{\!|\mathtt{k}|\!\}] : \mathtt{T} \S \Sigma$.

Case SENDCASE-R.

$$\frac{h(\tilde{\mathtt{k}}) = \overline{\mathtt{o}} \quad h(\mathtt{o}) = (\mathtt{C}, \_) \quad \mathtt{C} \Downarrow \{\mathtt{C}_1, \mathtt{C}_2\} = \mathtt{C}_i}{\mathscr{E}[\mathtt{k}.\mathtt{sendC}(\mathtt{o})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}], h \longrightarrow \mathscr{E}[\mathtt{e}_i], h[\tilde{\mathtt{k}} \mapsto \overline{\mathtt{o}} :: \mathtt{o}]}$$

By $h(\mathtt{o}) = (\mathtt{C}, \_)$ and $ag(\Gamma; h)$ we get that $\Gamma \vdash_{\mathtt{r}} \mathtt{o} : \mathtt{C} \S \emptyset$ using AXIOM-RT.

Since, by hypothesis $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{k}.\mathtt{sendC}(\mathtt{o})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}] : \mathtt{T} \S \Sigma$, by Lemma 6.2(1), we have that $\Gamma \vdash_{\mathtt{r}} \mathtt{k}.\mathtt{sendC}(\mathtt{o})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T}' \S \Sigma_1$ and $\Sigma = \Sigma_1.\Sigma_2$. From rule SENDC-RT we have that $\Sigma_1 = \{\mathtt{k} :! \{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}\}$ and $\Gamma \vdash_{\mathtt{r}} \mathtt{e}_i : \mathtt{T}' \S \{\mathtt{k} : \mathtt{t}_i\}$.

By Lemma 6.2(2), we have that $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{e}_i] : \mathtt{T} \S \Sigma'$, where $\Sigma' = \{\mathtt{k} : \mathtt{t}_i\}.\Sigma_2$.

From Definition 6.2 (LATER-3 and LATER-4) we conclude $\Sigma \sqsubseteq \Sigma'$.

Notice that the new heap $h[\tilde{\mathtt{k}} \mapsto \overline{\mathtt{o}} :: \mathtt{o}]$ still agrees with Γ since the only changes are about channels.

Case RECEIVECASE-R.

$$\frac{h(\mathtt{k}) = \mathtt{o} :: \overline{\mathtt{o}} \quad h(\mathtt{o}) = (\mathtt{C}, \_) \quad \mathtt{C} \Downarrow \{\mathtt{C}_1, \mathtt{C}_2\} = \mathtt{C}_i}{\mathscr{E}[\mathtt{k}.\mathtt{recvC}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}], h \longrightarrow \mathscr{E}[[\mathtt{o}/\mathtt{x}]\mathtt{e}_i], h[\mathtt{k} \mapsto \overline{\mathtt{o}}]}$$

By $h(\mathtt{o}) = (\mathtt{C}, \_)$ and $ag(\Gamma; h)$ we get that $\Gamma \vdash_{\mathtt{r}} \mathtt{o} : \mathtt{C} \S \emptyset$ using AXIOM-RT. Applying rule SUB-RT, we get $\Gamma \vdash_{\mathtt{r}} \mathtt{o} : \mathtt{C}_i \S \emptyset$.

Since, by hypothesis $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{k}.\mathtt{recvC}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\}] : \mathtt{T} \S \Sigma$, by Lemma 6.2(1), we have that $\Gamma \vdash_{\mathtt{r}} \mathtt{k}.\mathtt{recvC}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T}' \S \Sigma_1$ and $\Sigma = \Sigma_1.\Sigma_2$. From rule RECEIVEC-RT we have that $\Sigma_1 = \{\mathtt{k} :? \{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}\}$ and $\Gamma, \mathtt{x} : \mathtt{C}_i \vdash_{\mathtt{r}} \mathtt{e}_i : \mathtt{T}' \S \{\mathtt{k} : \mathtt{t}_i\}$.

By Lemma 6.3(1), we have that $\Gamma \vdash_{\mathtt{r}} [\mathtt{o}/\mathtt{x}]\mathtt{e}_i : \mathtt{T}' \S \{\mathtt{k} : \mathtt{t}_i\}$.

By Lemma 6.2(2), we have that $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[[\mathtt{o}/\mathtt{x}]\mathtt{e}_i] : \mathtt{T} \S \Sigma'$, where $\Sigma' = \{\mathtt{k} : \mathtt{t}_i\}.\Sigma_2$.

From Definition 6.2 (LATER-3 and LATER-4) we conclude $\Sigma \sqsubseteq \Sigma'$.

Notice that the new heap $h[\mathtt{k} \mapsto \overline{\mathtt{o}}]$ still agrees with Γ since the only changes are about channels.

Case SENDWHILE-R.

$$\mathscr{E}[\mathtt{k.sendW(e)\{C_1 \Rightarrow e_1 \,[]\, C_2 \Rightarrow e_2\}}],h \longrightarrow \mathscr{E}[\mathtt{k.sendC(e)\{C_1 \Rightarrow e'_1 \,[]\, C_2 \Rightarrow e'_2\}}],h$$

where $\mathtt{e}'_i = \mathtt{e}_i\lfloor \mathtt{k.sendW(e)\{C_1 \Rightarrow e_1 \,[]\, C_2 \Rightarrow e_2\}/cont}\rfloor$.
Since, by hypothesis $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{k.sendW(e)\{C_1 \Rightarrow e_1 \,[]\, C_2 \Rightarrow e_2\}}] : \mathtt{T} \fatsemi \Sigma$, by Lemma 6.2(1), we have that $\Gamma \vdash_{\mathtt{r}} \mathtt{k.sendW(e)\{C_1 \Rightarrow e_1 \,[]\, C_2 \Rightarrow e_2\}} : \mathtt{T}' \fatsemi \Sigma_1$ and $\Sigma = \Sigma_1.\Sigma_2$. From rule SENDW-RT we have that $\Sigma_1 = \{\mathtt{k} : \mu\alpha.!\{\mathtt{C_1} \Rightarrow [\alpha/\circledcirc]\mathtt{t}_1 \,[]\, \mathtt{C_2} \Rightarrow [\alpha/\circledcirc]\mathtt{t}_2\}\}$ and $\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{C_1} \vee \mathtt{C_2} \fatsemi \emptyset$ and $\Gamma \vdash_{\mathtt{r}} \mathtt{e}_i : \mathtt{T}' \fatsemi \{\mathtt{k} : \mathtt{t}_i\}$ and α fresh in $\mathtt{t}_1, \mathtt{t}_2$ and $\mathtt{T}' <: \mathtt{T}''$ for all $\mathtt{T}'' \in tc(\mathtt{e}_1) \cup tc(\mathtt{e}_2)$.
Let $\mathtt{t}'_i = [(\mu\alpha.!\{\mathtt{C_1} \Rightarrow [\alpha/\circledcirc]\mathtt{t}_1 \,[]\, \mathtt{C_2} \Rightarrow [\alpha/\circledcirc]\mathtt{t}_2\})/\circledcirc]\mathtt{t}_i$. By Lemma 6.3(2), we have that $\Gamma \vdash_{\mathtt{r}} \mathtt{e}'_i : \mathtt{T}' \fatsemi \{\mathtt{k} : \mathtt{t}'_i\}$.
By rule SENDC-RT, we get that

$$\Gamma \vdash_{\mathtt{r}} \mathtt{k.sendC(e)\{C_1 \Rightarrow e'_1 \,[]\, C_2 \Rightarrow e'_2\}} : \mathtt{T}' \fatsemi \mathtt{k} :!\{\mathtt{C_1} \Rightarrow \mathtt{t}'_1 \,[]\, \mathtt{C_2} \Rightarrow \mathtt{t}'_2\}.$$

By Lemma 6.2(2), we conclude $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{k.sendC(e)\{C_1 \Rightarrow e'_1 \,[]\, C_2 \Rightarrow e'_2\}}] : \mathtt{T} \fatsemi \Sigma'$, where $\Sigma' = \{\mathtt{k} :!\{\mathtt{C_1} \Rightarrow \mathtt{t}'_1 \,[]\, \mathtt{C_2} \Rightarrow \mathtt{t}'_2\}\}.\Sigma_2$ and $\Sigma \sqsubseteq \Sigma'$ by Definition 6.2 (LATER-1 and LATER-3), since we consider recursive types modulo fold/unfold.

Case RECEIVEWHILE-R.

$$\mathscr{E}[\mathtt{k.recvW(x)\{C_1 \Rightarrow e_1 \,[]\, C_2 \Rightarrow e_2\}}],h \longrightarrow \mathscr{E}[\mathtt{k.recvC(x)\{C_1 \Rightarrow e'_1 \,[]\, C_2 \Rightarrow e'_2\}}],h$$

where $\mathtt{e}'_i = \mathtt{e}_i\lfloor \mathtt{k.recvW(x)\{C_1 \Rightarrow e_1 \,[]\, C_2 \Rightarrow e_2\}/cont}\rfloor$.
Since, by hypothesis $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{k.recvW(x)\{C_1 \Rightarrow e_1 \,[]\, C_2 \Rightarrow e_2\}}] : \mathtt{T} \fatsemi \Sigma$, by Lemma 6.2(1), we have that $\Gamma \vdash_{\mathtt{r}} \mathtt{k.recvW(x)\{C_1 \Rightarrow e_1 \,[]\, C_2 \Rightarrow e_2\}} : \mathtt{T}' \fatsemi \Sigma_1$ and $\Sigma = \Sigma_1.\Sigma_2$. From rule RECEIVEW-RT we have that $\Sigma_1 = \{\mathtt{k} : \mu\alpha.?\{\mathtt{C_1} \Rightarrow [\alpha/\circledcirc]\mathtt{t}_1 \,[]\, \mathtt{C_2} \Rightarrow [\alpha/\circledcirc]\mathtt{t}_2\}\}$ and $\Gamma, \mathtt{x} : \mathtt{C}_i \vdash_{\mathtt{r}} \mathtt{e}_i : \mathtt{T}' \fatsemi \{\mathtt{k} : \mathtt{t}_i\}$, and α fresh in $\mathtt{t}_1, \mathtt{t}_2$ and $\mathtt{T}' <: \mathtt{T}''$ for all $\mathtt{T}'' \in tc(\mathtt{e}_1) \cup tc(\mathtt{e}_2)$.
Let $\mathtt{t}'_i = [(\mu\alpha.?\{\mathtt{C_1} \Rightarrow [\alpha/\circledcirc]\mathtt{t}_1 \,[]\, \mathtt{C_2} \Rightarrow [\alpha/\circledcirc]\mathtt{t}_2\})/\circledcirc]\mathtt{t}_i$. By Lemma 6.3(3), we have that $\Gamma, \mathtt{x} : \mathtt{C}_i \vdash_{\mathtt{r}} \mathtt{e}'_i : \mathtt{T}' \fatsemi \{\mathtt{k} : \mathtt{t}'_i\}$.
By rule RECEIVEC-RT, we get that

$$\Gamma \vdash_{\mathtt{r}} \mathtt{k.recvC(x)\{C_1 \Rightarrow e'_1 \,[]\, C_2 \Rightarrow e'_2\}} : \mathtt{T}' \fatsemi \mathtt{k} :?\{\mathtt{C_1} \Rightarrow \mathtt{t}'_1 \,[]\, \mathtt{C_2} \Rightarrow \mathtt{t}'_2\}.$$

By Lemma 6.2(2), we conclude $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{k.recvC(x)\{C_1 \Rightarrow e'_1 \,[]\, C_2 \Rightarrow e'_2\}}] : \mathtt{T} \fatsemi \Sigma'$, where $\Sigma' = \{\mathtt{k} :?\{\mathtt{C_1} \Rightarrow \mathtt{t}'_1 \,[]\, \mathtt{C_2} \Rightarrow \mathtt{t}'_2\}\}.\Sigma_2$ and $\Sigma \sqsubseteq \Sigma'$ by Definition 6.2 (LATER-1 and LATER-3), since we consider recursive types modulo fold/unfold.

The remaining cases easily follow from the induction hypothesis. □

Using the Subject Reduction theorem we show that expressions, which are obtained by reducing initial expressions, are typed from environments which agree with the current heap.

Corollary 6.1. If $\mathtt{e}$ is an initial expression and $\mathtt{e}, [] \longrightarrow^* \mathtt{e}' \mid\mid P, h$, then $\Gamma \vdash_{\mathtt{r}} \mathtt{e}' : \mathtt{T} \fatsemi \Sigma$ for some Γ, $\mathtt{T}$, Σ such that $ag(\Gamma, h)$.

Proof. The proof is by induction on $\longrightarrow^*$. The basic case is immediate by definition of initial expression. In the induction case by definition $\mathtt{e}, [] \longrightarrow^* \mathtt{e}' \mid\mid P, h$ means $\mathtt{e}, [] \longrightarrow^* \mathtt{e}_1 \mid\mid \mathtt{e}_2 \mid\mid \ldots \mid\mid \mathtt{e}_n, h'$ and either $\mathtt{e}_1, h' \longrightarrow \mathtt{e}', h$ and $P \equiv \mathtt{e}_2 \mid\mid \ldots \mid\mid \mathtt{e}_n$ or $\mathtt{e}_1, h' \longrightarrow \mathtt{e}' \mid\mid \mathtt{e}'', h$ and $P \equiv \mathtt{e}'' \mid\mid \mathtt{e}_2 \mid\mid \ldots \mid\mid \mathtt{e}_n$. By the induction hypothesis, $\mathtt{e}_1$ is well typed from a term environment

which agrees with h'. Therefore $\mathtt{e}'$ is well typed from a term environment which agrees with h by Theorem 6.1. □

6.2. *Type safety*

The run-time errors which our type system has to prevent are:

1 the selection of a field and the request of a session which do not belong to the class of the current object;
2 the creation of a pair of dual channels whose communication sequences do not perfectly match.

In particular, concerning the second point, we want to show that communications of well-typed sessions cannot be in a stuck situation. To this aim we have to study global properties of type preservation during the reduction of parallel threads, namely we need to take into account the objects in the queues associated to channels and their relations with the session types of the channels themselves.

In the following definition we extend the notion of duality between session types taking into account also the objects already sent by a thread, and waiting to be read by the thread which has the dual channel.

Definition 6.4. Let h be a heap, $\overline{\mathtt{o}}$ be a queue of objects in h and $\mathtt{t},\mathtt{t}'$ two session types. The relation $\mathtt{t} \ltimes_h^{\overline{\mathtt{o}}} \mathtt{t}'$ is defined by:

1 $\mathtt{t} \ltimes_h^{()} \mathtt{t}'$ if $\mathtt{t} \bowtie \mathtt{t}'$,
2 $\mathtt{t}_i.\mathtt{t}' \ltimes_h^{\overline{\mathtt{o}}::\mathtt{o}} \mathtt{t}''$ for $i \in \{1,2\}$ if $!\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}.\mathtt{t}' \ltimes_h^{\overline{\mathtt{o}}} \mathtt{t}''$ and $h(\mathtt{o}) = (\mathtt{C},\_)$ and $\mathtt{C} \Downarrow \{\mathtt{C}_1,\mathtt{C}_2\} = \mathtt{C}_i$.

Intuitively, the definition above describes an agreement between the session type $\mathtt{t}$ of a channel $\mathtt{k}$ and the session type $\mathtt{t}'$ of $\tilde{\mathtt{k}}$ after the objects $\overline{\mathtt{o}}$ have been put in the queue associated with $\tilde{\mathtt{k}}$ in h (recall that communication is asynchronous and that only one between the queues $h(\mathtt{k})$ and $h(\tilde{\mathtt{k}})$ can be nonempty). Thus, when the queue is empty (case (1) of the definition), $\mathtt{t}'$ and $\mathtt{t}$ agree if they are dual. When the queue is $\overline{\mathtt{o}} :: \mathtt{o}_i$ (case (2)), if the session type $\mathtt{t}''$ agrees with $!\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}.\mathtt{t}'$ after the objects $\overline{\mathtt{o}}$ have been put in the queue, then it also agrees with the type $\mathtt{t}_i.\mathtt{t}'$, where $\mathtt{t}_i$ is the session type of the branch obtained after putting in the queue the object $\mathtt{o}_i$.
For instance, $\mathtt{t}''$ agrees with $\mathtt{t}_1.\mathtt{t}'$ via the queue $\mathtt{"a"::true::3}$ (notation $\mathtt{t}_1.\mathtt{t}' \ltimes_h^{\mathtt{"a"::true::3}} \mathtt{t}''$) if it agrees with $!\{\mathtt{Int} \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{Object} \Rightarrow \mathtt{t}_2\}.\mathtt{t}'$ via the queue $\mathtt{"a"::true}$: indeed after branch selection (the sent value 3 is an $\mathtt{Int}$) the continuation of $!\{\mathtt{Int} \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{Object} \Rightarrow \mathtt{t}_2\}.\mathtt{t}'$ is $\mathtt{t}_1.\mathtt{t}'$.

The main lemma concerning the above relation says that if the type $\mathtt{t}$ of a channel $\mathtt{k}$ agrees with the type $?\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}.\mathtt{t}'$ of $\tilde{\mathtt{k}}$ when h maps $\tilde{\mathtt{k}}$ to the queue $\mathtt{o} :: \overline{\mathtt{o}}$, and $\mathtt{C} \Downarrow \{\mathtt{C}_1,\mathtt{C}_2\} = \mathtt{C}_i$, where $i \in \{1,2\}$ and $\mathtt{C}$ is the class of $\mathtt{o}$ in h, then $\mathtt{t}$ agrees with $\mathtt{t}_i.\mathtt{t}'$ when h maps $\tilde{\mathtt{k}}$ to the queue $\overline{\mathtt{o}}$.

Lemma 6.5. Let $\mathtt{t} \ltimes_h^{\mathtt{o}::\overline{\mathtt{o}}} ?\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}.\mathtt{t}'$, and $h(\mathtt{o}) = (\mathtt{C},\_)$, and $\mathtt{C} \Downarrow \{\mathtt{C}_1,\mathtt{C}_2\} = \mathtt{C}_i$, then $\mathtt{t} \ltimes_h^{\overline{\mathtt{o}}} \mathtt{t}_i.\mathtt{t}'$.

Proof. By induction on the length of $\overline{\mathtt{o}}$.

In the base case $\overline{\mathsf{o}} = ()$ the relation $\mathtt{t} \ltimes_h^{\mathsf{o}} ?\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}.\mathtt{t}'$ can only have been obtained by case (2) of Definition 6.4. So we have $\mathtt{t} = \mathtt{t}'_j.\mathtt{t}^*$ for some $\mathtt{t}'_j$ ($j \in \{1,2\}$) and $\mathtt{t}^*$ and $\mathtt{C} \Downarrow \{\mathtt{C}'_1, \mathtt{C}'_2\} = \mathtt{C}_j$ and $!\{\mathtt{C}'_1 \Rightarrow \mathtt{t}'_1 \,[]\, \mathtt{C}'_2 \Rightarrow \mathtt{t}'_2\}.\mathtt{t}^* \ltimes_h^{()} ?\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}.\mathtt{t}'$. By case (1) of Definition 6.4 we get $!\{\mathtt{C}'_1 \Rightarrow \mathtt{t}'_1 \,[]\, \mathtt{C}'_2 \Rightarrow \mathtt{t}'_2\}.\mathtt{t}^* \bowtie ?\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}.\mathtt{t}'$. From $\mathtt{C} \Downarrow \{\mathtt{C}_1, \mathtt{C}_2\} = \mathtt{C}_i$ and $\mathtt{C} \Downarrow \{\mathtt{C}'_1, \mathtt{C}'_2\} = \mathtt{C}_j$ we can derive either $\mathtt{C}_i \Downarrow \{\mathtt{C}'_1, \mathtt{C}'_2\} = \mathtt{C}_j$ or $\mathtt{C}_j \Downarrow \{\mathtt{C}_1, \mathtt{C}_2\} = \mathtt{C}_i$, which imply $\mathtt{t}_i \bowtie \mathtt{t}'_j$ and $\mathtt{t}^* \bowtie \mathtt{t}'$ by definition of duality. Therefore we conclude $\mathtt{t}_i.\mathtt{t}^* \bowtie \mathtt{t}'_j.\mathtt{t}'$ which gives $\mathtt{t}_i.\mathtt{t}^* \ltimes_h^{()} \mathtt{t}'_j.\mathtt{t}'$ by Definition 6.4(1).

For the induction case assume $\overline{\mathsf{o}} = \overline{\mathsf{o}'} :: \mathsf{o}^+$: thus the hypothesis becomes $\mathtt{t} \ltimes_h^{\mathsf{o}::\overline{\mathsf{o}'}::\mathsf{o}^+} ?\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}.\mathtt{t}'$. This relation can only have been obtained by case (2) of Definition 6.4. So we have $\mathtt{t} = \mathtt{t}^+_j.\mathtt{t}''$ for some $\mathtt{t}^+_j$ ($j \in \{1,2\}$) and $\mathtt{t}''$ and $h(\mathsf{o}^+) = (\mathtt{C}^+, \_)$ and $\mathtt{C}^+ \Downarrow \{\mathtt{C}^+_1, \mathtt{C}^+_2\} = \mathtt{C}^+_j$ and $!\{\mathtt{C}^+_1 \Rightarrow \mathtt{C}^+_2 \,[]\, \mathtt{t}^+_1 \Rightarrow \mathtt{t}^+_2\}.\mathtt{t}'' \ltimes_h^{\mathsf{o}::\overline{\mathsf{o}'}} ?\{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}.\mathtt{t}'$. By the induction hypothesis we have $\mathtt{C} \Downarrow \{\mathtt{C}_1, \mathtt{C}_2\} = \mathtt{C}_i$ and $!\{\mathtt{C}^+_1 \Rightarrow \mathtt{C}^+_2 \,[]\, \mathtt{t}^+_1 \Rightarrow \mathtt{t}^+_2\}.\mathtt{t}'' \ltimes_h^{\overline{\mathsf{o}'}} \mathtt{t}_i.\mathtt{t}'$. Applying again Definition 6.4(2) we get the result. □

We now extend the definition of *agreement* to session environments.

Definition 6.5.

1 The predicate $ag(\Sigma; h)$ is defined by:

$$ag(\Sigma;h) \quad \text{if} \quad \begin{cases} \mathtt{k} \in \mathsf{dom}(\Sigma) \;\Leftrightarrow\; \mathtt{k} \in \mathsf{dom}(h), \\ \forall \mathtt{k} \in \mathsf{dom}(\Sigma) : h(\mathtt{k}) = () \;\Rightarrow\; \Sigma(\mathtt{k}) \ltimes_h^{h(\tilde{\mathtt{k}})} \Sigma(\tilde{\mathtt{k}}). \end{cases}$$

2 $ag(\Gamma; \Sigma; h)$ if $ag(\Gamma; h)$ and $ag(\Sigma; h)$.

Then a session environment and a heap agree if:

— the same set of channels occurs in the session environment and in the heap,
— when the queue of a channel $\mathtt{k}$ is empty, then the queue of $\tilde{\mathtt{k}}$ relates the session type of $\mathtt{k}$ with the session type of $\tilde{\mathtt{k}}$.

A term environment, a session environment and a heap agree if both the heap with the standard environment and the heap with the session environment agree.

The following key lemma generalises Theorem 6.1, asserting that the above agreement is preserved under reduction of well-typed parallel threads.

Lemma 6.6 (Subject Reduction Generalisation). Let $\Gamma \vdash_{\mathtt{r}} \mathtt{e}_i : \mathtt{T}_i \,\S\, \Sigma_i$, $(1 \le i \le n)$, and assume $ag(\Gamma; \Sigma; h)$ where $\Sigma = \bigcup_{1 \le i \le n} \Sigma_i$. Then if

$$\mathtt{e}_1 \mid\mid \ldots \mid\mid \mathtt{e}_n, h \longrightarrow \mathtt{e}'_1 \mid\mid \ldots \mid\mid \mathtt{e}'_{n'}, h' \text{ where } 1 \le n \le n',$$

then there exist Γ' and Σ'_i such that $\Gamma' \vdash_{\mathtt{r}} \mathtt{e}'_i : \mathtt{T}_i \,\S\, \Sigma'_i$ $(1 \le i \le n')$ and $ag(\Gamma'; \Sigma'; h')$, where $\Sigma' = \bigcup_{1 \le i \le n'} \Sigma'_i$.

Proof. We have that, for some i $(1 \le i \le n)$, either $\mathtt{e}_i, h \longrightarrow \mathtt{e}'_i \mid\mid \mathtt{e}''_i, h'$ by an application of rule SESSREQ-R or $\mathtt{e}_i, h \longrightarrow \mathtt{e}'_i, h'$ by the application of any one of the other reduction rules. In the first case the proof follows immediately by Theorem 6.1(2) and Definition 6.5.

So let $\mathtt{e}_i, h \longrightarrow \mathtt{e}'_i, h'$. If this reduction has not been obtained by a communication rule the proof is trivial by Theorem 6.1(1). The interesting cases are when the reduction $\mathtt{e}_i, h \longrightarrow \mathtt{e}'_i, h'$ is

obtained by a communication rule. By Theorem 6.1 we immediately obtain $\Gamma' \vdash_{\mathtt{r}} \mathtt{e}'_i : \mathtt{T}_i \fatsemi \Sigma'_i$ and $ag(\Gamma';h')$, thus we only have to show $ag(\Sigma';h')$, which implies $ag(\Gamma';\Sigma';h')$.

Case SENDCASE-R. Assume $\mathtt{e}_i = \mathscr{E}[\mathtt{k}.\mathtt{sendC}(\mathtt{o})\{\mathtt{C}_1 \Rightarrow \mathtt{e}''_1 \,[\!]\, \mathtt{C}_2 \Rightarrow \mathtt{e}''_2\}]$. We have that

$$\mathscr{E}[\mathtt{k}.\mathtt{sendC}(\mathtt{o})\{\mathtt{C}_1 \Rightarrow \mathtt{e}''_1 \,[\!]\, \mathtt{C}_2 \Rightarrow \mathtt{e}''_2\}],\ h \quad\longrightarrow\quad \mathscr{E}[\mathtt{e}''_j],\ h' \quad \text{with } j \in \{1,2\}$$

where $h(\tilde{\mathtt{k}}) = \overline{\mathtt{o}}$, and $h(\mathtt{o}) = (\mathtt{C},\_)$, and $h' = h[\tilde{\mathtt{k}} \mapsto \overline{\mathtt{o}} :: \mathtt{o}]$, and $\mathtt{C} \Downarrow \{\mathtt{C}_1,\mathtt{C}_2\} = \mathtt{C}_j$.
Since $\Gamma \vdash_{\mathtt{r}} \mathtt{e}_i : \mathtt{T}_i \fatsemi \Sigma_i$, by the proof of the same case in Theorem 6.1 we get
$\Sigma_i = \{\mathtt{k} :! \{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[\!]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}\}.\Sigma''_i$ and $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{e}''_j] : \mathtt{T}_\fatsemi \Sigma'_i$ where $\Sigma'_i = \{\mathtt{k} : \mathtt{t}_j\}.\Sigma''_i$ for $j \in \{1,2\}$.
So we derive $\Sigma'(\mathtt{k}) \ltimes^{h'(\tilde{\mathtt{k}})}_{h'} \Sigma'(\tilde{\mathtt{k}})$ from $\Sigma(\mathtt{k}) \ltimes^{h(\tilde{\mathtt{k}})}_{h} \Sigma(\tilde{\mathtt{k}})$ by Definition 6.4(2), and we conclude $ag(\Sigma';h')$.

Case RECEIVECASE-R. Assume $\mathtt{e}_i = \mathscr{E}[\mathtt{k}.\mathtt{recvC}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}''_1 \,[\!]\, \mathtt{C}_2 \Rightarrow \mathtt{e}''_2\}]$. We have that

$$\mathscr{E}[\mathtt{k}.\mathtt{recvC}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}''_1 \,[\!]\, \mathtt{C}_2 \Rightarrow \mathtt{e}''_2\}],\ h \quad\longrightarrow\quad \mathscr{E}[[\mathtt{o}/\mathtt{x}]\mathtt{e}''_j],\ h' \quad \text{with } j \in \{1,2\}$$

where $h(\mathtt{k}) = \mathtt{o} :: \overline{\mathtt{o}}$, and $h(\mathtt{o}) = (\mathtt{C},\_)$, and $h' = h[\mathtt{k} \mapsto \overline{\mathtt{o}}]$, and $\mathtt{C} \Downarrow \{\mathtt{C}_1,\mathtt{C}_2\} = \mathtt{C}_j$.
Since $\Gamma \vdash_{\mathtt{r}} \mathtt{e}_i : \mathtt{T}_i \fatsemi \Sigma_i$, by the proof of the same case in Theorem 6.1 we get
$\Sigma_i = \{\mathtt{k} :? \{\mathtt{C}_1 \Rightarrow \mathtt{t}_1 \,[\!]\, \mathtt{C}_2 \Rightarrow \mathtt{t}_2\}\}.\Sigma''_i$ and $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[[\mathtt{o}/\mathtt{x}]\mathtt{e}''_j] : \mathtt{T}_\fatsemi \Sigma'_i$ where $\Sigma'_i = \{\mathtt{k} : \mathtt{t}_j\}.\Sigma''_i$ for $j \in \{1,2\}$. So we derive $\Sigma'(\tilde{\mathtt{k}}) \ltimes^{h'(\mathtt{k})}_{h'} \Sigma'(\mathtt{k})$ from $\Sigma(\tilde{\mathtt{k}}) \ltimes^{h(\mathtt{k})}_{h} \Sigma(\mathtt{k})$ by Lemma 6.5, and we conclude $ag(\Sigma';h')$.

□

It is handy to take into account the order in which communication and delegation redexes (see Section 4) which occur in the same expression are reduced. To this aim we introduce the "follows" relation between redexes.

Definition 6.6. Let $\mathtt{e}$ be an expression and $\mathtt{r}_1, \mathtt{r}_2$ be two different occurrences of communication or delegation redexes in $\mathtt{e}$. We say that $\mathtt{r}_2$ *follows* $\mathtt{r}_1$ in $\mathtt{e}$ if there is a subexpression $\mathtt{e}'$ of $\mathtt{e}$ such that $\mathtt{e}' = \mathscr{E}[\mathtt{r}_1]$ and $\mathtt{r}_2$ occurs in $\mathtt{e}'$.

Note that by definition of evaluation context $\mathtt{r}_1$ cannot be a subexpression of $\mathtt{r}_2$, while $\mathtt{r}_2$ can be a subexpression of $\mathtt{r}_1$.

It is easy to check that, if $\mathtt{r}_1$ and $\mathtt{r}_2$ are as in the previous definition, then $\mathtt{r}_1$ needs to be reduced before $\mathtt{r}_2$, since $\mathtt{r}_1$ occurs in the hole of an evaluation context $\mathscr{E}$, while $\mathtt{r}_2$ occurs elsewhere in the same expression.

We convene that all fresh channels created reducing parallel threads take successive indexes according to the order of creation, i.e. they are named $\mathtt{k}_0, \mathtt{k}_1, \ldots$. This means that if

$$P,h \longrightarrow^* Q,h' \longrightarrow^* Q',h''$$

and $\mathtt{k}_i$ is a channel created in the reduction $P,h \longrightarrow^* Q,h'$, and $\mathtt{k}_j$ is a channel created in the reduction $Q,h' \longrightarrow^* Q',h''$, then $i < j$.

The *subject* of a communication or delegation redex is the channel specified in its syntax on which the communication takes place. The *index of a communication or delegation redex* is the index of its subject.

The following crucial lemma states that a channel and its dual cannot occur in the same thread. Moreover it states that the order on indexes of communication and delegation redexes agrees with the "follows" relation between redexes.

Lemma 6.7. Let $\mathtt{e}$ be an initial expression and $\mathtt{e}, [] \longrightarrow^* \mathtt{e}_1 \mid\mid \ldots \mid\mid \mathtt{e}_n, h$. Then:

1 no expression $\mathtt{e}_i$ can contain occurrences of both $\mathtt{k}$ and $\tilde{\mathtt{k}}$ for some channel $\mathtt{k}$,

2 if $\mathtt{r}_1, \mathtt{r}_2$ are two different occurrences of communication or delegation redexes in $\mathtt{e}_i$ ($i \in \{1, \ldots, n\}$) and $\mathtt{r}_2$ follows $\mathtt{r}_1$, then the index of $\mathtt{r}_1$ is greater than or equal to the index of $\mathtt{r}_2$.

Proof.

1 Straightforward, noting that the channels $\mathtt{k}$ and $\tilde{\mathtt{k}}$ are introduced by the rule SESSREQ-R in two different parallel threads.

2 $\emptyset \vdash \mathtt{e} : \mathtt{T} \, ; \, \varepsilon$ implies that no channel occurs in $\mathtt{e}$ and so the property trivially holds. We now prove that the reduction preserves the property, namely if all the channels in the subexpressions of an expression are indexed in a not increasing order, starting from the redex to all the following redexes, in the sense of Definition 6.6, then after one step of reduction we get expressions that have the same property. The proof is by case analysis on the definition of $\longrightarrow$.

Case SESSREQ-R. We have that

$$\frac{h(\mathtt{o}) = (\mathtt{C}, \_) \qquad \mathit{sbody}(\mathtt{s}, \mathtt{C}) = \mathtt{e}'' \qquad \mathtt{k}, \tilde{\mathtt{k}} \notin h}{\mathscr{E}[\mathtt{o.s}\,\{\mathtt{e}'\}], h \longrightarrow \mathscr{E}[\mathtt{e}'\{\!|\mathtt{k}|\!\}] \mid\mid [\mathtt{o}/\mathtt{this}]\mathtt{e}''\{\!|\tilde{\mathtt{k}}|\!\}, h[\mathtt{k}, \tilde{\mathtt{k}} \mapsto ()]}$$

Let $\mathscr{E}[\mathtt{o.s}\,\{\mathtt{e}'\}]$ be an expression in which the desired property holds. After one step of reduction, in the expression $\mathtt{e}'\{\!|\mathtt{k}|\!\}$ the new channel $\mathtt{k}$ is the one with the highest index and no other channel occurs in it. Moreover all communication and delegation redexes occurring in $\mathscr{E}$ follow all communication and delegation redexes in $\mathtt{e}'\{\!|\mathtt{k}|\!\}$. Lastly, note that by the induction hypothesis the desired property holds for all communication and delegation redexes occurring in $\mathscr{E}$.
In parallel we have the expression $[\mathtt{o}/\mathtt{this}]\mathtt{e}''\{\!|\tilde{\mathtt{k}}|\!\}$, where $\mathtt{e}''$ is a session body, so the only channel in this expression is $\tilde{\mathtt{k}}$. Then this reduction rule preserves the property.

Case SESSDEL-R. We have that

$$\frac{h(\mathtt{o}) = (\mathtt{C}, \_) \qquad \mathit{sbody}(\mathtt{s}, \mathtt{C}) = \mathtt{e}'}{\mathscr{E}[\mathtt{o} \bullet \mathtt{s}\,\{\mathtt{k}\}], h \longrightarrow \mathscr{E}[[\mathtt{o}/\mathtt{this}]\mathtt{e}'\{\!|\mathtt{k}|\!\}], h}$$

Let $\mathscr{E}[\mathtt{o} \bullet \mathtt{s}\,\{\mathtt{k}\}]$ be an expression in which the desired property holds. Since $\mathtt{o} \bullet \mathtt{s}\,\{\mathtt{k}\}$ is the redex, then $\mathtt{k}$ is the channel with the highest index. After one step of reduction, $[\mathtt{o}/\mathtt{this}]\mathtt{e}'\{\!|\mathtt{k}|\!\}$ is the first expression to be reduced next, and $\mathtt{k}$ is still the only channel which occurs in it.

Case SENDCASE-R. We have that

$$\frac{h(\tilde{\mathtt{k}}) = \overline{\mathtt{o}} \qquad h(\mathtt{o}) = (\mathtt{C}, \_) \qquad \mathtt{C} \Downarrow \{\mathtt{C}_1, \mathtt{C}_2\} = \mathtt{C}_i}{\mathscr{E}[\mathtt{k}.\mathtt{sendC}(\mathtt{o})\{\mathtt{C}_1 \Rightarrow \mathtt{e}'_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}'_2\}], h \longrightarrow \mathscr{E}[\mathtt{e}'_j], h[\tilde{\mathtt{k}} \mapsto \overline{\mathtt{o}} :: \mathtt{o}]}$$

If the expression $\mathscr{E}[\mathtt{k}.\mathtt{sendC}(\mathtt{o})\{\mathtt{C}_1 \Rightarrow \mathtt{e}'_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}'_2\}]$ is an expression in which the desired property holds, then $\mathtt{k}$ is the channel with the highest index. The channel $\mathtt{k}$ is the only channel which occurs in the expressions $\mathtt{e}'_1, \mathtt{e}'_2$. Then, after one step of reduction the

expression $\mathtt{e}'_j$ can contain only the channel $\mathtt{k}$, that is the one with the highest index, or it can contain no channel, then the property still holds.

Cases RECEIVECASE-R, SENDWHILE-R and RECEIVEWHILE-R. The proof is similar to the previous one.

In all the remaining cases no channel is introduced or modified, therefore the property is trivially preserved.

□

The above lemma is a technical step to prove the *deadlock freedom* property for communication expressions. Indeed, it is easy to verify that well-typed sending redexes always reduce, as well as while-receiving redexes. Then the crucial case is when we obtain a parallel composition of case-receiving redexes: in the following lemma we prove that these receiving actions are not stuck, since their expectations match the values on the channel queue.

Lemma 6.8 (Deadlock Freedom). Let $\mathtt{e}$ be an initial expression and

$$\mathtt{e}, [\,] \longrightarrow^* \mathtt{o}_1 \mid\mid \ldots \mid\mid \mathtt{o}_m \mid\mid \mathtt{e}_1 \mid\mid \ldots \mid\mid \mathtt{e}_n, h,$$

such that $m \geq 0$ and for all i $(1 \leq i \leq n)$ $\mathtt{e}_i = \mathscr{E}_i[\mathtt{r}_i]$, where $\mathscr{E}_i$ is an evaluation context and $\mathtt{r}_i$ is a case-receiving redex. Then there is i $(1 \leq i \leq n)$, such that $\mathtt{e}_i, h \longrightarrow P, h'$ for some P, h'.

Proof. By Corollary 6.1, each $\mathtt{e}_i$ is well typed from a term environment Γ which agrees with h.

Let j be the highest among the indexes of the channels occurring in $\mathtt{e}_1 \mid ... \mid \mathtt{e}_n$.

If both $\mathtt{k}_j$ and $\tilde{\mathtt{k}}_j$ occur in $\mathtt{e}_1 \mid ... \mid \mathtt{e}_n$, then by Lemma 6.7(1) they occur in two different expressions, let them be $\mathtt{e}_p$ and $\mathtt{e}_q$ with $1 \leq p \neq q \leq n$. By Lemma 6.7(2) the subjects of the two redexes $\mathtt{r}_p$ and $\mathtt{r}_q$ are the channels $\mathtt{k}_j$ and $\tilde{\mathtt{k}}_j$. Moreover we must have that $\Sigma_p(\mathtt{k}_j)$, $\Sigma_q(\tilde{\mathtt{k}}_j)$ are of the forms $?\{\mathtt{D}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{D}_2 \Rightarrow \mathtt{t}_2\}.\mathtt{t}$, $?\{\mathtt{D}'_1 \Rightarrow \mathtt{t}'_1 \,[]\, \mathtt{D}'_2 \Rightarrow \mathtt{t}'_2\}.\mathtt{t}'$, since $\mathtt{r}_p$ and $\mathtt{r}_q$ are case-receiving redexes. If $h(\mathtt{k}_j)$ is not empty, then let $h(\mathtt{k}_j) = \mathtt{o} :: \overline{\mathtt{o}'}$, and by Lemma 6.5 $h(\mathtt{o}) = (\mathtt{C}, \_)$ and $\mathtt{C} \Downarrow \{\mathtt{D}_1, \mathtt{D}_2\}$ is defined, so $\mathtt{r}_p$ can perform a RECEIVECASE-R step against the hypothesis. Similarly if $h(\tilde{\mathtt{k}}_j)$ is not empty. Otherwise, if both $h(\mathtt{k}_j)$ and $h(\tilde{\mathtt{k}}_j)$ are empty, then by Lemma 6.6 we get $ag(\Sigma; h)$, where Σ is the session environment of $\mathtt{e}_1 \mid ... \mid \mathtt{e}_n$. This implies $\Sigma(\mathtt{k}) \ltimes_h^{()} \Sigma(\tilde{\mathtt{k}})$ by Definition 6.5 and then $\Sigma_q(\tilde{\mathtt{k}}_j) \bowtie \Sigma_p(\mathtt{k}_j)$ by Definition 6.4(1). But this is impossible since $\Sigma_p(\mathtt{k}_j)$ and $\Sigma_q(\tilde{\mathtt{k}}_j)$ are of the forms $?\{\mathtt{D}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathtt{D}_2 \Rightarrow \mathtt{t}_2\}.\mathtt{t}$, $?\{\mathtt{D}'_1 \Rightarrow \mathtt{t}'_1 \,[]\, \mathtt{D}'_2 \Rightarrow \mathtt{t}'_2\}.\mathtt{t}'$.

If only $\mathtt{k}_j$ occurs in $\mathtt{e}_1 \mid ... \mid \mathtt{e}_n$, then we must have $\Sigma(\mathtt{k}_j) \neq \varepsilon$, $\Sigma(\tilde{\mathtt{k}}_j) = \varepsilon$. From $ag(\Sigma; h)$, by Definition 6.5, we get that $\Sigma(\tilde{\mathtt{k}}_j) = \varepsilon$ implies $h(\tilde{\mathtt{k}}_j) = ()$ and then $\varepsilon \ltimes_h^{h(\mathtt{k}_j)} \Sigma(\mathtt{k}_j)$. We conclude that $h(\mathtt{k}_j)$ is not empty, so we can proceed as before. □

Theorem 6.2 (Type safety). If $\mathtt{e}$ is an initial expression and $\mathtt{e}, [\,] \longrightarrow^* \mathtt{e}_1 \mid\mid \ldots \mid\mid \mathtt{e}_n, h$, then one of the following conditions holds:

— there is i $(1 \leq i \leq n)$, such that $\mathtt{e}_i, h \longrightarrow P, h'$ for some P, h',
— for all i $(1 \leq i \leq n)$, $\mathtt{e}_i$ is an object.

Proof. By Proposition 6.1 $\mathtt{e}$ is closed and channel-complete, so by Proposition 4.2 each $\mathtt{e}_i$ is closed and channel-complete. Therefore, by Proposition 4.1, either $\mathtt{e}_i$ is an object identifier or $\mathtt{e}_i = \mathscr{E}_i[\mathtt{r}_i]$, for some evaluation context $\mathscr{E}_i$ and some redex $\mathtt{r}_i$. If $\mathtt{e}_i = \mathscr{E}_i[\mathtt{r}_i]$ then, by Corollary 6.1,

$\mathtt{e}_i$ can be typed from a term environment Γ which agrees with h, hence $\mathtt{r}_i$ can be typed from Γ too, by Lemma 6.2(1).

If some $\mathtt{r}_i$ is of one of the shapes $\mathtt{o;e'}$ or $\mathtt{new\ C}(\overline{\mathtt{o}})$ or $\mathtt{k.sendC}(\mathtt{o})\{\mathtt{C} \Rightarrow \mathtt{e} \,[]\, \mathtt{C} \Rightarrow \mathtt{e}\}$ or $\mathtt{k.sendW}(\mathtt{e})\{\mathtt{C} \Rightarrow \mathtt{e} \,[]\, \mathtt{C} \Rightarrow \mathtt{e}\}$ or $\mathtt{k.recvW}(\mathtt{x})\{\mathtt{C} \Rightarrow \mathtt{e} \,[]\, \mathtt{C} \Rightarrow \mathtt{e}\}$, then it is immediate to verify that $\mathtt{e}_i$ reduces.

Otherwise, let some $\mathtt{r}_i$ be of one of the shapes $\mathtt{o.f}$ or $\mathtt{o.f} := \mathtt{o'}$ or $\mathtt{o.s}\{\mathtt{e'}\}$ or $\mathtt{o} \bullet \mathtt{s}\{\mathtt{k}\}$. Since an object identifier cannot occur in an initial expression, then the run-time expression $\mathtt{o}$ has been obtained by reducing $\mathtt{new\ C}(\overline{\mathtt{o}})$ for some $\mathtt{C}$, $\overline{\mathtt{f} : \mathtt{o}}$, which implies $h(\mathtt{o}) = (\mathtt{C}, \overline{\mathtt{f} : \mathtt{o}})$ by rule (NewC-R). By Definition 6.3(2) this implies $\Gamma(\mathtt{o}) = \mathtt{C}$. If $\mathtt{r}_i = \mathtt{o.f}$, then rule (FLD-RT) has been applied with a premise $\Gamma \vdash_{\mathtt{r}} \mathtt{o} : \mathtt{T} \fatsemi \emptyset$ for some $\mathtt{T}$ such that $\mathtt{C} <: \mathtt{T}$ and $\mathtt{f} \in \mathit{fields}(\mathtt{T})$, therefore $\mathtt{f} \in \mathit{fields}(\mathtt{C})$. If $\mathtt{r}_i = \mathtt{o.s}\{\ldots\}$, then rule (SESSREQ-RT) has been applied with a premise $\Gamma \vdash_{\mathtt{r}} \mathtt{o} : \mathtt{T} \fatsemi \emptyset$ for some $\mathtt{T}$ such that $\mathtt{C} <: \mathtt{T}$ and $\mathit{stype}(\mathtt{s},\mathtt{T})$ is defined. Therefore also $\mathit{stype}(\mathtt{s},\mathtt{C})$ is defined, and then $\mathit{sbody}(\mathtt{s},\mathtt{C})$ is defined. Similarly one can show that $\mathit{sbody}(\mathtt{s},\mathtt{C})$ is defined when $\mathtt{r}_i = \mathtt{o} \bullet \mathtt{s}\{\ldots\}$. Then, in all the cases above, we can conclude that $\mathtt{e}_i$ reduces.

The only remaining alternative when all $\mathtt{r}_i$ are case-receiving redexes follows from Lemma 6.8. □

7. Session Type Reconstruction

The type system presented in Figure 9 derives session types for expressions assuming that all session declarations are decorated with explicit session types; moreover, expressions can have many types due to the presence of the subsumption rule. In this section, we present an inference algorithm (Figure 13) which

i) gives an expression its minimal type;
ii) calculates the constraints that must be satisfied in order to reconstruct the related session type (which is unique as stated in Proposition 5.1).

Thus the programmer is no longer responsible for declaring the session types.

We define an *inference class table ICT* as a class table in which each session declaration $\mathtt{s}$, in each class $\mathtt{C}$, is decorated by the *session-in-class variable* $\chi^{\mathtt{s}}_{\mathtt{C}}$ representing the session type that will be inferred by the algorithm.

Then we extend the syntax of session types to *session type schemes* in order to include session-in-class variables:

$$\theta ::= \mathtt{t} \mid \chi^{\mathtt{s}}_{\mathtt{C}} \mid \theta.\theta \mid \dagger\{\mathtt{C}_1 \Rightarrow \theta \,[]\, \mathtt{C}_2 \Rightarrow \theta\} \mid \mu\alpha.\dagger\{\mathtt{C}_1 \Rightarrow \theta \,[]\, \mathtt{C}_2 \Rightarrow \theta\}$$

If CT is a class table, we denote by CT^{-} the inference class table obtained by replacing in CT the declared session type of any session $\mathtt{s}$ in any class $\mathtt{C}$ by $\chi^{\mathtt{s}}_{\mathtt{C}}$.

In order to reconstruct session types of session declarations, we use two kinds of constraints. A set of *equality* (and *disequality*) constraints, denoted by $\mathscr{C}$, will collect assertions of the shape $\chi^{\mathtt{s}}_{\mathtt{C}} = \theta$ and $\chi^{\mathtt{s}}_{\mathtt{C}} \neq \varepsilon$. A set of *duality* constraints, denoted by $\mathscr{D}$, will collect assertions of the shape $\chi^{\mathtt{s}}_{\mathtt{C}} \bowtie \theta$.

The constraint-based type inference system is presented in Figure 13. We notice that, if a session-in-class variable $\chi^{\mathtt{s}}_{\mathtt{C}}$ occurs in a session type that is inferred for an expression, then $\chi^{\mathtt{s}}_{\mathtt{C}}$ has been introduced by rule SESSDEL-T-I; therefore the related set of constraints must contain

$$\textsc{Axiom-T-I}\quad \Gamma \vdash_{\mathrm{I}} \mathtt{z} : \Gamma(\mathtt{z}) \fatsemi \varepsilon \rhd \emptyset, \emptyset \qquad \textsc{Cont-T-I}\quad \Gamma \vdash_{\mathrm{I}} \mathtt{cont}^{\mathtt{T}} : \mathtt{T} \fatsemi \circledcirc \rhd \emptyset, \emptyset$$

$$\textsc{NewC-T-I}\ \frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{T}\,\mathtt{f}} \qquad \Gamma \vdash_{\mathrm{I}} \mathtt{e}_i : \mathtt{T}'_i \fatsemi \varepsilon \rhd \mathscr{C}_i, \mathscr{D}_i \qquad \mathtt{T}'_i <: \mathtt{T}_i}{\Gamma \vdash_{\mathrm{I}} \mathtt{new}\,\mathtt{C}(\overline{\mathtt{e}}) : \mathtt{C} \fatsemi \varepsilon \rhd \bigcup_i \mathscr{C}_i, \bigcup_i \mathscr{D}_i} \qquad \textsc{Fld-T-I}\ \frac{\Gamma \vdash_{\mathrm{I}} \mathtt{e} : \mathtt{T} \fatsemi \theta \rhd \mathscr{C}, \mathscr{D}}{\Gamma \vdash_{\mathrm{I}} \mathtt{e.f} : \mathit{ftype}_r(\mathtt{f},\mathtt{T}) \fatsemi \theta \rhd \mathscr{C}, \mathscr{D}}$$

$$\textsc{Seq-T-I}\ \frac{\Gamma \vdash_{\mathrm{I}} \mathtt{e} : \mathtt{T} \fatsemi \theta \rhd \mathscr{C}, \mathscr{D} \qquad \Gamma \vdash_{\mathrm{I}} \mathtt{e}' : \mathtt{T}' \fatsemi \theta' \rhd \mathscr{C}', \mathscr{D}'}{\Gamma \vdash_{\mathrm{I}} \mathtt{e};\mathtt{e}' : \mathtt{T}' \fatsemi \theta.\theta' \rhd \mathscr{C} \cup \mathscr{C}', \mathscr{D} \cup \mathscr{D}'}$$

$$\textsc{FldAss-T-I}\ \frac{\Gamma \vdash_{\mathrm{I}} \mathtt{e} : \mathtt{T} \fatsemi \theta \rhd \mathscr{C}, \mathscr{D} \qquad \Gamma \vdash_{\mathrm{I}} \mathtt{e}' : \mathtt{T}' \fatsemi \theta' \rhd \mathscr{C}', \mathscr{D}' \qquad \mathtt{T}' <: \mathit{ftype}_w(\mathtt{f},\mathtt{T})}{\Gamma \vdash_{\mathrm{I}} \mathtt{e.f} := \mathtt{e}' : \mathit{ftype}_r(\mathtt{f},\mathtt{T}) \fatsemi \theta.\theta' \rhd \mathscr{C} \cup \mathscr{C}', \mathscr{D} \cup \mathscr{D}'}$$

$$\textsc{SessReq-T-I}\ \frac{\Gamma \vdash_{\mathrm{I}} \mathtt{e} : \mathtt{C}_1 \vee \cdots \vee \mathtt{C}_n \fatsemi \theta \rhd \mathscr{C}, \mathscr{D} \qquad \Gamma \vdash_{\mathrm{I}} \mathtt{e}' : \mathtt{T}' \fatsemi \theta' \rhd \mathscr{C}', \mathscr{D}' \qquad \mathscr{D}'' = \mathscr{D} \cup \mathscr{D}' \cup \{\chi^{\mathtt{s}}_{\mathtt{C}_i} \bowtie \theta' | i \in \{1,\ldots,n\}\}}{\Gamma \vdash_{\mathrm{I}} \mathtt{e.s}\{\mathtt{e}'\} : \mathtt{T}' \fatsemi \theta \rhd \mathscr{C} \cup \mathscr{C}', \mathscr{D}''}$$

$$\textsc{SessDel-T-I}\ \frac{\Gamma \vdash_{\mathrm{I}} \mathtt{e} : \mathtt{C}_1 \vee \cdots \vee \mathtt{C}_n \fatsemi \theta \rhd \mathscr{C}, \mathscr{D} \qquad \mathit{rtype}(\mathtt{s}, \mathtt{C}_1 \vee \cdots \vee \mathtt{C}_n) = \mathtt{T} \qquad \mathscr{C}' = \mathscr{C} \cup \{\chi^{\mathtt{s}}_{\mathtt{C}_1} \neq \varepsilon\} \cup \{\chi^{\mathtt{s}}_{\mathtt{C}_i} = \chi^{\mathtt{s}}_{\mathtt{C}_j} | i \neq j \in \{1,\ldots,n\}\}}{\Gamma \vdash_{\mathrm{I}} \mathtt{e} \bullet \mathtt{s}\{\} : \mathtt{T} \fatsemi \theta.\chi^{\mathtt{s}}_{\mathtt{C}_1} \rhd \mathscr{C}', \mathscr{D}}$$

$$\textsc{SendC-T-I}\ \frac{\Gamma \vdash_{\mathrm{I}} \mathtt{e} : \mathtt{T} \fatsemi \varepsilon \rhd \mathscr{C}, \mathscr{D} \qquad \Gamma \vdash_{\mathrm{I}} \mathtt{e}_i : \mathtt{T}_i \fatsemi \theta_i \rhd \mathscr{C}_i, \mathscr{D}_i \qquad \mathtt{T} <: \mathtt{C}_1 \vee \mathtt{C}_2}{\Gamma \vdash_{\mathrm{I}} \mathtt{sendC}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T}_1 \vee \mathtt{T}_2 \fatsemi !\{\mathtt{C}_1 \Rightarrow \theta_1 \,[]\, \mathtt{C}_2 \Rightarrow \theta_2\} \rhd \mathscr{C} \cup \mathscr{C}_1 \cup \mathscr{C}_2, \mathscr{D} \cup \mathscr{D}_1 \cup \mathscr{D}_2}$$

$$\textsc{ReceiveC-T-I}\ \frac{\Gamma, \mathtt{x} : \mathtt{C}_i \vdash_{\mathrm{I}} \mathtt{e}_i : \mathtt{T}_i \fatsemi \theta_i \rhd \mathscr{C}_i, \mathscr{D}_i}{\Gamma \vdash_{\mathrm{I}} \mathtt{recvC}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T}_1 \vee \mathtt{T}_2 \fatsemi ?\{\mathtt{C}_1 \Rightarrow \theta_1 \,[]\, \mathtt{C}_2 \Rightarrow \theta_2\} \rhd \mathscr{C}_1 \cup \mathscr{C}_2, \mathscr{D}_1 \cup \mathscr{D}_2}$$

$$\textsc{SendW-T-I}\ \frac{\Gamma \vdash_{\mathrm{I}} \mathtt{e} : \mathtt{T} \fatsemi \varepsilon \rhd \mathscr{C}, \mathscr{D} \qquad \Gamma \vdash_{\mathrm{I}} \mathtt{e}_i : \mathtt{T}_i \fatsemi \theta_i \rhd \mathscr{C}_i, \mathscr{D}_i \qquad \alpha \text{ fresh in } \theta_1, \theta_2 \qquad \mathtt{T}_1 \vee \mathtt{T}_2 <: \mathtt{T}' \qquad \forall \mathtt{T}' \in tc(\mathtt{e}_1) \cup tc(\mathtt{e}_2) \qquad \mathtt{T} <: \mathtt{C}_1 \vee \mathtt{C}_2}{\Gamma \vdash_{\mathrm{I}} \mathtt{sendW}(\mathtt{e})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T}_1 \vee \mathtt{T}_2 \fatsemi \mu\alpha.!\{\mathtt{C}_1 \Rightarrow [\alpha/\circledcirc]\theta_1 \,[]\, \mathtt{C}_2 \Rightarrow [\alpha/\circledcirc]\theta_2\} \rhd \mathscr{C} \cup \mathscr{C}_1 \cup \mathscr{C}_2, \mathscr{D} \cup \mathscr{D}_1 \cup \mathscr{D}_2}$$

$$\textsc{ReceiveW-T-I}\ \frac{\Gamma, \mathtt{x} : \mathtt{C}_i \vdash_{\mathrm{I}} \mathtt{e}_i : \mathtt{T}_i \fatsemi \theta_i \rhd \mathscr{C}_i, \mathscr{D}_i \qquad \alpha \text{ fresh in } \theta_1, \theta_2 \qquad \mathtt{T}_1 \vee \mathtt{T}_2 <: \mathtt{T} \qquad \forall \mathtt{T} \in tc(\mathtt{e}_1) \cup tc(\mathtt{e}_2)}{\Gamma \vdash_{\mathrm{I}} \mathtt{recvW}(\mathtt{x})\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T}_1 \vee \mathtt{T}_2 \fatsemi \mu\alpha.?\{\mathtt{C}_1 \Rightarrow [\alpha/\circledcirc]\theta_1 \,[]\, \mathtt{C}_2 \Rightarrow [\alpha/\circledcirc]\theta_2\} \rhd \mathscr{C}_1 \cup \mathscr{C}_2, \mathscr{D}_1 \cup \mathscr{D}_2}$$

Fig. 13. Constraint-based Typing Rules for Channel Free Expressions.

$\chi^{\mathtt{s}}_{\mathtt{C}} \neq \varepsilon$. Then, no derived session type can be equated to ε by a substitution which satisfies the set of constraints. For this reason we write explicitly ε, when required, in the antecedents of the inference rules.

$$\textsc{Sess-WF-I}\quad \frac{\{\texttt{this}:\texttt{C}\}\vdash_{\mathrm{I}} \texttt{e}:\texttt{T}\fatsemi\theta \triangleright \mathscr{C}',\mathscr{D} \qquad \mathscr{C}=\mathscr{C}'\cup\{\chi^{\texttt{s}}_{\texttt{C}}=\theta\} \qquad \circledcirc \text{ does not appear in } \theta}{\texttt{T}\,\chi^{\texttt{s}}_{\texttt{C}}\,\texttt{s}\,\{\ \texttt{e}\ \}\ \texttt{ok in C with } \mathscr{C},\mathscr{D}}$$

$$\textsc{Class-WF-I}\quad \frac{\texttt{D ok} \qquad \texttt{S}_i\ \texttt{ok in C with } \mathscr{C}_i,\mathscr{D}_i}{\texttt{class C} \lhd \texttt{D}\ \{\ \overline{\texttt{T f}};\ \overline{\texttt{S}}\ \}\ \texttt{ok with } \bigcup_i \mathscr{C}_i, \bigcup_i \mathscr{D}_i}$$

Fig. 14. Well-formed Inference Class Tables.

In the communication rules the resulting minimal type is the union of the types of the two branches, i.e. their least supertype.

Rules for well-formedness of session and class declarations are in Figure 14. The declaration of a session $\texttt{s}$ in a class $\texttt{C}$ is well formed under the constraints $\mathscr{C}$ and $\mathscr{D}$ if the body $\texttt{e}$ is well typed under constraints $\mathscr{C}'$ and $\mathscr{D}$. The set $\mathscr{C}$ includes the constraints collected typing the body $\texttt{e}$ ($\mathscr{C}'$) and the equation $\chi^{\texttt{s}}_{\texttt{C}}=\theta$ which assigns to the session variable $\chi^{\texttt{s}}_{\texttt{C}}$ the session type scheme θ representing the communications performed in the body $\texttt{e}$. As for rule SESS-WF, the condition that $\circledcirc$ does not appear in θ is justified by the fact that $\circledcirc$ has no dual type, so sessions whose bodies would be typed with types containing $\circledcirc$ would be useless.

The well-formedness of a class declaration is checked under the union of the constraints collected checking the well-formedness of session declarations in $\texttt{C}$.

We define the *set of constraints of an inference class table ICT* as the pair $\langle \bigcup_{i\in I}\mathscr{C}_i ; \bigcup_{i\in I}\mathscr{D}_i\rangle$, where $\texttt{C}_i$ for $i\in I$ is the set of classes defined in *ICT* and $\texttt{class C}_i\ \ldots\ \texttt{ok}$ with $\mathscr{D}_i,\mathscr{C}_i$.

8. Properties of the Constraint-Based Typing

In this section we prove that the constraint typing rules of Figure 13 are sound and complete w.r.t. the typing rules of Figure 9.

Indeed, given an inference class table *ICT*, which is well formed under constraints $\langle\mathscr{C};\mathscr{D}\rangle$, if σ is a substitution which satisfies $\mathscr{C}$ and $\mathscr{D}$, then $\sigma(ICT)$ gives a well-formed class table according to the type derivation $\vdash$ (Soundness).

Conversely, for any well-formed class table *CT*, the corresponding inference class table CT^{-} results to be well formed under constraints $\langle\mathscr{C};\mathscr{D}\rangle$ such that there is a unique substitution σ which satisfies $\mathscr{C}$ and $\mathscr{D}$ (Completeness). Furthermore, we prove that $\sigma(CT^{-})=CT$.

Definition 8.1 (Type substitution). A type substitution σ is a finite mapping from sessions type variables to session types. The application of a substitution to a session type scheme is defined as follows:

$$\sigma(\texttt{t})=\texttt{t}$$
$$\sigma(\chi^{\texttt{s}}_{\texttt{C}})=\begin{cases}\texttt{t} & \text{if } (\chi^{\texttt{s}}_{\texttt{C}}\mapsto\texttt{t})\in\sigma\\ \chi^{\texttt{s}}_{\texttt{C}} & \text{if } \chi^{\texttt{s}}_{\texttt{C}}\notin \mathsf{dom}(\sigma)\end{cases}$$
$$\sigma(\theta.\theta')=\sigma(\theta).\sigma(\theta')$$
$$\sigma(\dagger\{\texttt{C}_1\Rightarrow\theta_1\,[]\,\texttt{C}_2\Rightarrow\theta_2\})=\dagger\{\texttt{C}_1\Rightarrow\sigma(\theta_1)\,[]\,\texttt{C}_2\Rightarrow\sigma(\theta_2)\}$$
$$\sigma(\mu\alpha.\dagger\{\texttt{C}_1\Rightarrow\theta_1\,[]\,\texttt{C}_2\Rightarrow\theta_2\})=\mu\alpha.\dagger\{\texttt{C}_1\Rightarrow\sigma(\theta_1)\,[]\,\texttt{C}_2\Rightarrow\sigma(\theta_2)\}.$$

Substitutions on inference class tables are defined as expected.

In the soundness property formulation it suffices to consider expressions which occur in class tables.

Theorem 8.1 (Soundness). Let *ICT* be an inference class table with set of constraints $\langle \mathscr{C}'; \mathscr{D}' \rangle$. If $\Gamma \vdash_{\mathrm{I}} \mathtt{e} : \mathtt{T} \fatsemi \theta \rhd \mathscr{C}, \mathscr{D}$ using *ICT* is such that $\mathscr{C} \subseteq \mathscr{C}'$ and $\mathscr{D} \subseteq \mathscr{D}'$ and σ is a substitution that satisfies $\mathscr{C}'$ and $\mathscr{D}'$, then $\Gamma \vdash \mathtt{e} : \mathtt{T} \fatsemi \sigma(\theta)$ using the class table $\sigma(ICT)$.

Proof. By induction on the type derivation of $\Gamma \vdash_{\mathrm{I}} \mathtt{e} : \mathtt{T} \fatsemi \theta \rhd \mathscr{C}, \mathscr{D}$, with a case analysis on the final rule. We only consider the most interesting cases.

Notice that σ satisfies $\mathscr{C}'$ and $\mathscr{D}'$, and then $\sigma(ICT)$ is a class table and $\sigma(\theta)$ is a session type.

Case FLDASS-T-I. We have that $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_1.\mathtt{f} := \mathtt{e}_2 : \mathtt{T} \fatsemi \theta \rhd \mathscr{C}, \mathscr{D}$. From rule FLDASS-T-I we have that $\mathtt{T} = \mathit{ftype}_r(\mathtt{f}, \mathtt{T}_1)$, $\mathscr{C} = \mathscr{C}_1 \cup \mathscr{C}_2$, $\mathscr{D} = \mathscr{D}_1 \cup \mathscr{D}_2$, $\theta = \theta_1.\theta_2$, $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_1 : \mathtt{T}_1 \fatsemi \theta_1 \rhd \mathscr{C}_1, \mathscr{D}_1$, and $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_2 : \mathtt{T}_2 \fatsemi \theta_2 \rhd \mathscr{C}_2, \mathscr{D}_2$, for some $\mathscr{C}_1, \mathscr{C}_2, \mathscr{D}_1, \mathscr{D}_2, \mathtt{T}_1$ and some $\mathtt{T}_2$ such that $\mathtt{T}_2 <: \mathit{ftype}_w(\mathtt{f}, \mathtt{T}_1)$. Since $\mathscr{C}_1 \subseteq \mathscr{C} \subseteq \mathscr{C}'$, $\mathscr{C}_2 \subseteq \mathscr{C} \subseteq \mathscr{C}'$, $\mathscr{D}_1 \subseteq \mathscr{D} \subseteq \mathscr{D}'$, and $\mathscr{D}_2 \subseteq \mathscr{D} \subseteq \mathscr{D}'$, by the induction hypothesis, $\Gamma \vdash \mathtt{e}_1 : \mathtt{T}_1 \fatsemi \sigma(\theta_1)$ and $\Gamma \vdash \mathtt{e}_2 : \mathtt{T}_2 \fatsemi \sigma(\theta_2)$ using the class table $\sigma(ICT)$. By applying rules SUB-T (since $\mathit{ftype}_w(\mathtt{f}, \mathtt{T}_1) <: \mathit{ftype}_r(\mathtt{f}, \mathtt{T}_1)$ by definition) and FLDASS-T, we get the result: $\Gamma \vdash \mathtt{e}_1.\mathtt{f} := \mathtt{e}_2 : \mathtt{T} \fatsemi \sigma(\theta)$ using the class table $\sigma(ICT)$.

Case SESSREQ-T-I. We have that $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_1.\mathtt{s}\{\mathtt{e}_2\} : \mathtt{T} \fatsemi \theta \rhd \mathscr{C}, \mathscr{D}$. From rule SESSREQ-T-I we have that $\mathscr{C} = \mathscr{C}_1 \cup \mathscr{C}_2$, $\mathscr{D} = \mathscr{D}_1 \cup \mathscr{D}_2 \cup \{\chi^{\mathtt{s}}_{\mathtt{C}} \bowtie \theta' | \mathtt{C} \in \mathtt{T}'\}$, $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_1 : \mathtt{T}' \fatsemi \theta \rhd \mathscr{C}_1, \mathscr{D}_1$ and $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_2 : \mathtt{T} \fatsemi \theta' \rhd \mathscr{C}_2, \mathscr{D}_2$, for some $\mathscr{C}_1, \mathscr{C}_2, \mathscr{D}_1, \mathscr{D}_2, \mathtt{T}'$. Since $\mathscr{C}_1 \subseteq \mathscr{C} \subseteq \mathscr{C}'$, $\mathscr{C}_2 \subseteq \mathscr{C} \subseteq \mathscr{C}'$, $\mathscr{D}_1 \subseteq \mathscr{D} \subseteq \mathscr{D}'$, and $\mathscr{D}_2 \subseteq \mathscr{D} \subseteq \mathscr{D}'$, by the induction hypothesis $\Gamma \vdash \mathtt{e}_1 : \mathtt{T}' \fatsemi \sigma(\theta)$ and $\Gamma \vdash \mathtt{e}_2 : \mathtt{T} \fatsemi \sigma(\theta')$. Moreover, the fact that σ satisfies $\mathscr{D}$, implies that $\sigma(\chi^{\mathtt{s}}_{\mathtt{C}}) \bowtie \sigma(\theta')$, for all $\mathtt{C} \in \mathtt{T}'$, and $\{\sigma(\chi^{\mathtt{s}}_{\mathtt{C}}) \,|\, \mathtt{C} \in \mathtt{T}'\} = \mathit{stype}(\mathtt{s}, \mathtt{T}')$, using the class table $\sigma(ICT)$. By applying rule SESSREQ-T we get the result: $\Gamma \vdash \mathtt{e}_1.\mathtt{s}\{\mathtt{e}_2\} : \mathtt{T} \fatsemi \sigma(\theta)$ using the class table $\sigma(ICT)$.

Case SESSDEL-T-I. We have that $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_0 \bullet \mathtt{s}\{\} : \mathtt{T} \fatsemi \theta \rhd \mathscr{C}, \mathscr{D}$. From rule SESSDEL-T-I we have that $\mathscr{C} = \mathscr{C}_1 \cup \{\chi^{\mathtt{s}}_{\mathtt{C}_1} \neq \varepsilon\} \cup \{\chi^{\mathtt{s}}_{\mathtt{C}_i} = \chi^{\mathtt{s}}_{\mathtt{C}_j} | i \neq j \in \{1, \ldots, n\}\}$, $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_0 : \mathtt{T}' \fatsemi \theta \rhd \mathscr{C}_1, \mathscr{D}$ and $\mathit{rtype}(\mathtt{s}, \mathtt{T}')$=T, and $\mathtt{T}' = \mathtt{C}_1 \vee \cdots \vee \mathtt{C}_n$, for some $\mathscr{C}_1, \mathtt{T}', \mathtt{C}_1, \cdots, \mathtt{C}_n$. Since $\mathscr{C}_1 \subseteq \mathscr{C} \subseteq \mathscr{C}'$, by the induction hypothesis $\Gamma \vdash \mathtt{e}_0 : \mathtt{T}' \fatsemi \sigma(\theta)$ and $\mathit{stype}(\mathtt{s}, \mathtt{T}') = \{\sigma(\chi^{\mathtt{s}}_{\mathtt{C}_1})\}$, with $\sigma(\chi^{\mathtt{s}}_{\mathtt{C}_1}) \neq \varepsilon$, using the class table $\sigma(ICT)$. By applying rule SESSDEL-T we get the result: $\Gamma \vdash \mathtt{e}_0 \bullet \mathtt{s}\{\} : \mathtt{T} \fatsemi \sigma(\theta)$ using the class table $\sigma(ICT)$.

Case SENDC-T-I. We have that $\Gamma \vdash_{\mathrm{I}} \mathtt{sendC}(\mathtt{e}_0)\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T}_1 \vee \mathtt{T}_2 \fatsemi \theta \rhd \mathscr{C}, \mathscr{D}$. From rule SENDC-T-I we have that $\theta = !\{\mathtt{C}_1 \Rightarrow \theta_1 \,[]\, \mathtt{C}_2 \Rightarrow \theta_2\}$, $\mathscr{C} = \mathscr{C}_1 \cup \mathscr{C}_2 \cup \mathscr{C}_3$, $\mathscr{D} = \mathscr{D}_1 \cup \mathscr{D}_2 \cup \mathscr{D}_3$ and $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_i : \mathtt{T}_i \fatsemi \theta_i \rhd \mathscr{C}_i, \mathscr{D}_i$ ($i \in \{1,2\}$) for some $\theta_1, \theta_2, \mathscr{C}_1, \mathscr{C}_2, \mathscr{C}_3, \mathscr{D}_1, \mathscr{D}_2, \mathscr{D}_3$. Moreover $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_0 : \mathtt{T}' \fatsemi \varepsilon \rhd \mathscr{C}_3, \mathscr{D}_3$ for some $\mathtt{T}' <: \mathtt{C}_1 \vee \mathtt{C}_2$. Since $\mathscr{C}_j \subseteq \mathscr{C} \subseteq \mathscr{C}'$, $\mathscr{D}_j \subseteq \mathscr{D} \subseteq \mathscr{D}'$, for $j \in \{1,2,3\}$, by the induction hypothesis $\Gamma \vdash \mathtt{e}_0 : \mathtt{T}' \fatsemi \varepsilon$ and $\Gamma \vdash \mathtt{e}_i : \mathtt{T}_i \fatsemi \sigma(\theta_i)$ using the class table $\sigma(ICT)$. By applying rule SUB-T, we get $\Gamma \vdash \mathtt{e}_0 : \mathtt{C}_1 \vee \mathtt{C}_2 \fatsemi \varepsilon$ and $\Gamma \vdash \mathtt{e}_i : \mathtt{T}_1 \vee \mathtt{T}_2 \fatsemi \sigma(\theta_i)$. Then SENDC-T applies, and we obtain $\Gamma \vdash \mathtt{sendC}(\mathtt{e}_0)\{\mathtt{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T}_1 \vee \mathtt{T}_2 \fatsemi \sigma(\theta)$ using the class table $\sigma(ICT)$.

Cases RECEIVEC-T-I, SENDW-T-I, RECEIVEW-T-I are similar.

□

Theorem 8.2 (Completeness). Let *CT* be a well formed class table and σ be a substitution such that $\{\sigma(\chi^{\mathtt{s}}_{\mathtt{C}})\} = \mathit{stype}(\mathtt{s}, \mathtt{C})$, for any $\mathtt{s}, \mathtt{C} \in CT$.

For any expression $\mathtt{e}$, if $\Gamma \vdash \mathtt{e} : \mathtt{T} \fatsemi \mathtt{t}$ using *CT*, then

i) $\Gamma \vdash_{\mathrm{I}} \mathtt{e} : \mathtt{T}' \fatsemi \theta \rhd \mathscr{C}, \mathscr{D}$ using CT^{-};
ii) $\mathtt{T}' <: \mathtt{T}$;
iii) σ satisfies $\mathscr{C}$ and $\mathscr{D}$;
iv) $\sigma(\theta) = \mathtt{t}$;

for some $\mathscr{C}, \mathscr{D}, \mathtt{T}'$.

Proof. By induction on the type derivation of $\Gamma \vdash \mathtt{e} : \mathtt{T} \fatsemi \mathtt{t}$, with a case analysis on the final rule. We only consider the most interesting cases.

Case FLDASS-T. We have that $\Gamma \vdash \mathtt{e}_1.\mathtt{f} := \mathtt{e}_2 : \mathtt{T} \fatsemi \mathtt{t}$. From rule FLDASS-T we have that $\mathtt{T} = \mathit{ftype}_r(\mathtt{f}, \mathtt{T}_1)$, $\mathtt{t} = \mathtt{t}_1.\mathtt{t}_2$, $\Gamma \vdash \mathtt{e}_1 : \mathtt{T}_1 \fatsemi \mathtt{t}_1$, and $\Gamma \vdash \mathtt{e}_2 : \mathit{ftype}_w(\mathtt{f}, \mathtt{T}_1) \fatsemi \mathtt{t}_2$ for some $\mathtt{T}_1$, $\mathtt{t}_1$, $\mathtt{t}_2$. By the induction hypothesis, we have:

1) $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_1 : \mathtt{T}'_1 \fatsemi \theta_1 \rhd \mathscr{C}_1, \mathscr{D}_1$ and $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_2 : \mathtt{T}_2 \fatsemi \theta_2 \rhd \mathscr{C}_2, \mathscr{D}_2$,

2) σ satisfies $\mathscr{C}_1$, $\mathscr{C}_2$, $\mathscr{D}_1$, $\mathscr{D}_2$, and $\sigma(\theta_1) = \mathtt{t}_1$ and $\sigma(\theta_2) = \mathtt{t}_2$,

for some $\mathtt{T}'_1 <: \mathtt{T}_1$, $\mathtt{T}_2 <: \mathit{ftype}_w(\mathtt{f}, \mathtt{T}_1)$, θ_1, θ_2, $\mathscr{C}_1$, $\mathscr{C}_2$,$\mathscr{D}_1$, and $\mathscr{D}_2$.
The condition $\mathtt{T}_2 <: \mathit{ftype}_w(\mathtt{f}, \mathtt{T}'_1)$ holds by $\mathtt{T}_2 <: \mathit{ftype}_w(\mathtt{f}, \mathtt{T}_1)$, since $\mathtt{T}'_1 <: \mathtt{T}_1$ implies $\mathit{ftype}_w(\mathtt{f}, \mathtt{T}_1) <: \mathit{ftype}_w(\mathtt{f}, \mathtt{T}'_1)$ by definition of ftype_w. Therefore we can apply rule FLDASS-T-I to 1) getting $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_1.\mathtt{f} := \mathtt{e}_2 : \mathit{ftype}_r(\mathtt{f}, \mathtt{T}'_1) \fatsemi \theta_1.\theta_2 \rhd \mathscr{C}_1 \cup \mathscr{C}_2, \mathscr{D}_1 \cup \mathscr{D}_2$. Note that $\mathtt{T}'_1 <: \mathtt{T}_1$ implies $\mathit{ftype}_r(\mathtt{f}, \mathtt{T}'_1) <: \mathit{ftype}_r(\mathtt{f}, \mathtt{T}_1)$ by definition of ftype_r. From 2) we conclude that σ satisfies $\mathscr{C}_1 \cup \mathscr{C}_2$ and $\mathscr{D}_1 \cup \mathscr{D}_2$, and that $\sigma(\theta_1.\theta_2) = \mathtt{t}$.

Case SESSREQ-T. We have that $\Gamma \vdash \mathtt{e}_1.\mathtt{s}\{\mathtt{e}_2\} : \mathtt{T} \fatsemi \mathtt{t}$. From rule SESSREQ-T we have that $\Gamma \vdash \mathtt{e}_1 : \mathtt{T}_1 \fatsemi \mathtt{t}$, $\Gamma \vdash \mathtt{e}_2 : \mathtt{T} \fatsemi \mathtt{t}_2$ and $\mathtt{t}_2 \bowtie \mathtt{t}'$, for some $\mathtt{T}_1$, $\mathtt{t}_2$ and for all $\mathtt{t}'$ such that $\mathtt{t}' \in \mathit{stype}(\mathtt{s}, \mathtt{T}_1)$. By the induction hypothesis, we have:

1) $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_1 : \mathtt{T}'_1 \fatsemi \theta_1 \rhd \mathscr{C}_1, \mathscr{D}_1$ and $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_2 : \mathtt{T}_2 \fatsemi \theta_2 \rhd \mathscr{C}_2, \mathscr{D}_2$,

2) σ satisfies $\mathscr{C}_1$, $\mathscr{C}_2$, $\mathscr{D}_1$, $\mathscr{D}_2$, and $\sigma(\theta_1) = \mathtt{t}$ and $\sigma(\theta_2) = \mathtt{t}_2$,

for some $\mathtt{T}'_1 <: \mathtt{T}_1$, $\mathtt{T}_2 <: \mathtt{T}$, θ_1, θ_2, $\mathscr{C}_2$, $\mathscr{C}_2$, $\mathscr{D}_1$, $\mathscr{D}_2$.
Let $\mathtt{T}_1 = \mathtt{C}_1 \vee \ldots \vee \mathtt{C}_n$. From rule SESSREQ-T-I and 1) we have that $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_1.\mathtt{s}\{\mathtt{e}_2\} : \mathtt{T}_2 \fatsemi \theta_1 \rhd \mathscr{C}, \mathscr{D}$, where $\mathscr{C} = \mathscr{C}_1 \cup \mathscr{C}_2$ and $\mathscr{D} = \mathscr{D}_1 \cup \mathscr{D}_2 \cup \{\chi^{\mathtt{s}}_{\mathtt{C}_i} \bowtie \theta_2 | i \in \{1,\ldots,n\}\}$. Since by hypothesis $\{\sigma(\chi^{\mathtt{s}}_{\mathtt{C}})\} = \mathit{stype}(\mathtt{s}, \mathtt{C})$ for all $\mathtt{s}, \mathtt{C} \in CT$, and by definition $\mathit{stype}(\mathtt{s}, \mathtt{T}_1) = \bigcup_{i \in \{1,\ldots,n\}} \mathit{stype}(\mathtt{s}, \mathtt{C}_i)$, then by 2) the condition $\mathtt{t}_2 \bowtie \mathtt{t}'$, for all $\mathtt{t}' \in \mathit{stype}(\mathtt{s}, \mathtt{T}_1)$, implies $\sigma(\chi^{\mathtt{s}}_{\mathtt{C}_i}) \bowtie \sigma(\theta_2)$, for all $i \in \{1,\ldots,n\}$. Then from 2) we conclude that σ satisfies $\mathscr{C}$ and $\mathscr{D}$.

Case SESSDEL-T. We have $\Gamma \vdash \mathtt{e}_0 \bullet \mathtt{s}\{\} : \mathtt{T} \fatsemi \mathtt{t}$. From rule SESSDEL-T we have that $\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 \fatsemi \mathtt{t}_0$, $\mathit{stype}(\mathtt{s}, \mathtt{T}_0) = \{\mathtt{t}'\}$, $\mathtt{t}' \neq \varepsilon$, $\mathtt{t} = \mathtt{t}_0.\mathtt{t}'$ and $\mathit{rtype}(\mathtt{s}, \mathtt{T}_0) = \mathtt{T}$. By the induction hypothesis, we have that

1) $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_0 : \mathtt{T}'_0 \fatsemi \theta_0 \rhd \mathscr{C}_0, \mathscr{D}$,

2) σ satisfies $\mathscr{C}_0$ and $\mathscr{D}$ and $\sigma(\theta_0) = \mathtt{t}_0$,

for some $\mathtt{T}'_0 <: \mathtt{T}_0$, θ_0, $\mathscr{C}_0$, $\mathscr{D}$.
Let $\mathtt{T}_0 = \mathtt{C}_1 \vee \ldots \vee \mathtt{C}_n$. From rule SESSDEL-T-I and 1) we have $\Gamma \vdash_{\mathrm{I}} \mathtt{e}_0 \bullet \mathtt{s}\{\} : \mathit{rtype}(\mathtt{s}, \mathtt{T}'_0) \fatsemi \theta_0.\chi^{\mathtt{s}}_{\mathtt{C}_1} \rhd \mathscr{C}, \mathscr{D}$, where $\mathscr{C} = \mathscr{C}_0 \cup \{\chi^{\mathtt{s}}_{\mathtt{C}_1} \neq \varepsilon\} \cup \{\chi^{\mathtt{s}}_{\mathtt{C}_i} = \chi^{\mathtt{s}}_{\mathtt{C}_j} | i \neq j \in \{1,\ldots,n\}\}$. Since $\mathit{stype}(\mathtt{s}, \mathtt{T}_0) = \{\mathtt{t}'\}$ implies $\mathit{stype}(\mathtt{s}, \mathtt{C}_i) = \{\mathtt{t}'\}$ for $i \in \{1,\ldots,n\}$ and by hypothesis

$\{\sigma(\chi^{\mathsf{s}}_{\mathsf{C}_i})\} = \mathit{stype}(\mathsf{s},\mathsf{C}_i)$ we get $\sigma(\chi^{\mathsf{s}}_{\mathsf{C}_i}) = \mathtt{t}'$. Therefore $\sigma(\theta.\chi^{\mathsf{s}}_{\mathsf{C}_1}) = \mathtt{t}_0.\mathtt{t}'$ and σ satisfies $\mathscr{C}$. From $\mathtt{T}'_0 <: \mathtt{T}_0$ we have $\mathit{rtype}(\mathsf{s},\mathtt{T}'_0) <: \mathit{rtype}(\mathsf{s},\mathtt{T}_0)$, then we obtain the result.

Case SENDC-T. We have $\Gamma \vdash \mathtt{sendC}(\mathtt{e}_0)\{\mathsf{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathsf{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T} \fatsemi \mathtt{t}$. From rule SENDC-T we have that $\mathtt{t} = !\{\mathsf{C}_1 \Rightarrow \mathtt{t}_1 \,[]\, \mathsf{C}_2 \Rightarrow \mathtt{t}_2\}$, and $\Gamma \vdash \mathtt{e}_0 : \mathsf{C}_1 \vee \mathsf{C}_2 \fatsemi \varepsilon$, and $\Gamma \vdash \mathtt{e}_i : \mathtt{T} \fatsemi \mathtt{t}_i$ for $i \in \{1,2\}$. By the induction hypothesis:

1) $\Gamma \vdash_1 \mathtt{e}_0 : \mathtt{T}' \fatsemi \varepsilon \rhd \mathscr{C}, \mathscr{D}$,

2) $\Gamma \vdash_1 \mathtt{e}_i : \mathtt{T}_i \fatsemi \theta_i \rhd \mathscr{C}_i, \mathscr{D}_i$,

3) σ satisfies $\mathscr{C}$, $\mathscr{D}$, $\mathscr{C}_i$, $\mathscr{D}_i$ and $\sigma(\theta_i) = \mathtt{t}_i$,

for some $\mathtt{T}' <: \mathsf{C}_1 \vee \mathsf{C}_2$, $\mathscr{C}$, $\mathscr{D}$, $\mathtt{T}_i <: \mathtt{T}$, θ_i, $\mathscr{C}_i$, and $\mathscr{D}_i$ with $i \in \{1,2\}$.
We can apply rule SENDC-T-I to obtain $\Gamma \vdash_1 \mathtt{sendC}(\mathtt{e}_0)\{\mathsf{C}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathsf{C}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{T}_1 \vee \mathtt{T}_2 \fatsemi !\{\mathsf{C}_1 \Rightarrow \theta_1 \,[]\, \mathsf{C}_2 \Rightarrow \theta_2\} \rhd \mathscr{C} \cup \mathscr{C}_1 \cup \mathscr{C}_2, \mathscr{D} \cup \mathscr{D}_1 \cup \mathscr{D}_2$. Since $\mathtt{T}_1 <: \mathtt{T}$ and $\mathtt{T}_2 <: \mathtt{T}$, we get $\mathtt{T}_1 \vee \mathtt{T}_2 <: \mathtt{T}$. From 3) we conclude that $\sigma(!\{\mathsf{C}_1 \Rightarrow \theta_1 \,[]\, \mathsf{C}_2 \Rightarrow \theta_2\}) = \mathtt{t}$ and σ satisfies $\mathscr{D} \cup \mathscr{D}_1 \cup \mathscr{D}_2$, $\mathscr{C} \cup \mathscr{C}_1 \cup \mathscr{C}_2$.

Cases RECEIVEC-T-I, SENDW-T-I, RECEIVEW-T-I are similar.

□

It is interesting to notice that we do not need to consider principal solutions as in the standard approach ((Pierce, 2002), Chapter 22). The reason is that the classes of exchanged objects are explicit in communication expressions.

Corollary 8.1 (Uniqueness of the solution). Let CT be a class table, and CT^{-} be the corresponding inference class table with constraints ($\mathscr{C}$,$\mathscr{D}$). Let $\sigma(CT^{-}) = CT$. For any substitution σ' that satisfies ($\mathscr{C}$,$\mathscr{D}$) and such that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ we get $\sigma' = \sigma$.

Proof. Let us suppose ad absurdum that $\sigma' \neq \sigma$, i.e. $\sigma'(CT^{-}) = CT' \neq CT$. The only difference between CT' and CT concerns the session types declared in the sessions definitions. This contradicts Proposition 5.1 and rule SESS-WF. □

Summarising, when read from bottom to top, the constraint typing rules determine an algorithm which calculates the constraints that must be satisfied in order for a class table ICT to be well formed. If a solution exists then it is the (unique) substitution σ which verifies both $\mathscr{C}$ and $\mathscr{D}$. The procedure for finding this substitution σ consists of two steps. First we apply a standard unification algorithm on first-order type expressions (see Chapter 22 of (Pierce, 2002)) to solve equality constraints of $\mathscr{C}$. Then we verify that σ satisfies all duality constraints of $\mathscr{D}$. If this procedure succeeds, then σ gives the session types which decorate all session declarations in such a way that $\sigma(ICT)$ is well formed.

9. Related work

Union types have been shown useful for enhancing the flexibility of subtyping in various settings: for functional languages (Barbanera et al., 1995; Frisch et al., 2008), for object-oriented languages (Igarashi and Nagira, 2007), for languages manipulating semi-structured data (Gapeyev and Pierce, 2003) and for the π-calculus (Castagna et al., 2008; Castagna et al., 2009).

It is interesting to compare $\mathscr{F}\mathsf{SAM}^{\vee}$ with FJ$\vee$, an extension of FJ with union types, proposed by Igarashi and Nagira in (Igarashi and Nagira, 2007). They define union types as in the

present paper: the essential difference is that they have traditional methods instead of sessions. The method signatures are of the shape $\overline{\mathtt{T}} \to \mathtt{T}$. The method type lookup function applied to a method name $\mathtt{m}$ and to a type $\mathtt{T}$ gives a set of method signatures, i.e. all the signatures of $\mathtt{m}$ in the classes which build $\mathtt{T}$. This is similar to our *stype* function, which returns a set of session types. The rule of method call checks that the types of the parameters agree with all the signatures found by the method type lookup function for the type of the object. Also our rule SESSREQ-T requires the session type of the co-body be dual to all the session types returned by the *stype* function. It is easy to check that the encoding of methods by sessions sketched at the end of Section 2 extends without changes to methods with union types.

Session types have been first introduced to model communication protocols between π-calculus processes (Honda, 1993; Takeuchi et al., 1994; Honda et al., 1998). They have been made more expressive by enriching them with correspondence assertions (Bonelli et al., 2005), subtyping (Gay and Hole, 2005), bounded polymorphism (Gay, 2008), higher-order processes (Mostrous and Yoshida, 2007; Mostrous and Yoshida, 2009), exceptions (Carbone et al., 2008b), concurrent constraints (Coppo and Dezani-Ciancaglini, 2009), and safer by assuring deadlock-freedom (Dezani-Ciancaglini et al., 2008; Bettini et al., 2008b). Session types have also been extended to multi-party communications (Bonelli and Compagnoni, 2008; Carbone et al., 2008a), with action permutations (Honda et al., 2009), design by contracts (Bocchi et al., 2010), dependent types for parametricity (Yoshida et al., 2010), upper bounds on buffer sizes (Deniélou and Yoshida, 2010) and access/information flow control (Capecchi et al., 2010a; Capecchi et al., 2011). Session types have been developed also for CORBA (Vallecillo et al., 2002), for functional languages (Gay et al., 2003; Vasconcelos et al., 2006; Bhargavan et al., 2009), for boxed ambients (Garralda et al., 2006), for the W3C standard description language for Web Services CDL (Carbone et al., 2007; Web Services Choreography Working Group, 2002; Sparkes, 2006; Honda et al., 2007), and for object-oriented programming languages.

The remaining of this section is devoted to the literature on session types in the object-oriented paradigm.

The papers (Dezani-Ciancaglini et al., 2005; Dezani-Ciancaglini et al., 2006; Coppo et al., 2007; Dezani-Ciancaglini et al., 2007; Dezani-Ciancaglini et al., 2009) discuss a multi-threaded object-oriented calculus augmented with session primitives, which supports session names as parameters of methods, spawning, iterative sessions and delegation.

The language Sing# (Fähndrich et al., 2006) is a variant of C# which combines session types with ownership types (Clarke et al., 2001), supports message-based communication via a designed heap area (shared memory), and allows interfaces between OS-modules to be described as message passing conversations. CoreSing# (Bono et al., 2011) is a core calculus inspired by the main features of Sing#. CoreSing# is equipped with a type system which uses session types and a novel form of ownership types to ensure the absence of communication errors, memory faults, and memory leaks in a communication model based on copyless message passing.

SJ (Hu et al., 2008) is an extension of Java with syntax for session types and structured communication operations. The main features of SJ are asynchronous message passing, delegation, session subtyping, interleaving, class downloading, and failure handling. (Hu et al., 2010) presents an extension of SJ which allows type-safe event-driven session programming.

(Gay et al., 2010) formalises a core distributed class-based object-oriented language with a static type system that combines session-typed channels and a form of typestates. Each class

definition has a session type which specifies the possible sequences of method calls. Channels can be stored in object fields, and separated methods implement parts of sessions. The availability of methods depends on the state of objects.

The amalgamation of the notion of session-based communication with that of object-oriented programming was first developed in (Drossopoulou et al., 2007). Characteristic of this design is that channel names are only generated at run time, and as a consequence only delegation of a session to another session within the same thread is expressible. Since the delegating and the delegated sessions can have different objects as receivers, this delegation is related for this respect to the delegation of method execution in object based calculi (Lieberman, 1986). $\mathscr{F}\mathsf{SAM}^{\vee}$ extends the calculus of (Drossopoulou et al., 2007) with union types and with a cleaner and simpler typing and operational semantics, since delegation in (Drossopoulou et al., 2007) requires ad hoc run time constructors. In (Capecchi et al., 2009) generic types are added to a language/calculus based on the approach of (Drossopoulou et al., 2007); we claim that union types fit better than generic types our communication primitives based on classes of exchanged objects. We think that the present type reconstruction cannot be easily adapted to the session types of (Capecchi et al., 2009) by intrinsic difficulties of the type inference with generic types. Giachino in (Giachino, 2009) presents an extension of $\mathscr{F}\mathsf{SAM}^{\vee}$ with intersection and negation types which allows a service-oriented interpretation of session overloading.

10. Conclusion

The core language $\mathscr{F}\mathsf{SAM}^{\vee}$, firstly presented in (Bettini et al., 2008a), showed how the addition of union types to an object oriented language with session types enhances flexibility.

In this paper we presented the full formalisation of the language and we proved that the language is type safe. Moreover, we presented an inference algorithm which gives an expression its minimal type and calculates the constraints that must be satisfied in order to reconstruct the related (unique) session type for each session declaration.

The language $\mathscr{F}\mathsf{SAM}^{\vee}$can be also viewed as a kernel proposal for generalising the standard notion of session-less methods in the object-oriented framework, where method-call interaction between two objects is limited to the initial sending of argument values for parameters. Once the syntax of the expressions is extended to send/receive operations, a method definition can include a sequence of interactions. The typechecking will be responsible for deriving session types for methods, so determining the appropriate evaluation rule to be used for method invocation: an empty session type will cause a standard semantics, while a non-empty one will use the evaluation rules defined in the present paper.

The amalgamation of communication centred and object oriented programming, as it has been developed in (Drossopoulou et al., 2007; Capecchi et al., 2009) and in the present paper, does not allow to express naturally some common patterns of concurrent programming. Session nesting is a strong limitation in the programming design, as for example the only way of having a "forwarder" is by creating a new session for each forwarded message. Our restricted delegation does not allow to write in a straightforward way a server that does load-balancing by delegation to worker threads. We plan to remove these drawbacks presumably by adding explicit channels and to extend our approach in various directions. In particular we plan to integrate this approach with multi-party session communication (Carbone et al., 2008a; Bettini et al., 2008b), with access

and information flow control (Capecchi et al., 2010a; Capecchi et al., 2011) and with exception handling (Capecchi et al., 2010b).

We plan to develop a prototype implementation of a language based on the approach presented in this paper; a possible tool for the implementation of the run-time system of our language could be IMC, `http://imc-fi.sourceforge.net`, a Java framework for implementing network applications, which provides reusable mechanisms to deal with the implementation of communication protocols. Indeed, IMC has already been used for implementing the run-time system of calculi with session-based communication primitives (Bettini et al., 2008c). This would also allow us to embedd our type system for session types in a distributed setting; we do not see crucial issues when transposing our session type setting to a distributed context, since our approach, as stated in the Introduction, is agnostic w.r.t. to the remaining aspects of the language. Of course, our session types do not deal with network failures, but only with the correctness of the communication protocols.

Acknowledgements We thank the anonymous referees for comments and suggestions, which significantly contributed to improving the earlier draft.

References

Barbanera, F., Dezani-Ciancaglini, M., and de'Liguoro, U. (1995). Intersection and Union Types: Syntax and Semantics. *Information and Computation*, 119:202–230.

Bettini, L., Capecchi, S., Dezani-Ciancaglini, M., Giachino, E., and Venneri, B. (2008a). Session and Union Types for Object Oriented Programming. In *Concurrency, Graphs and Models*, volume 5065 of *LNCS*, pages 659–680. Springer.

Bettini, L., Coppo, M., D'Antoni, L., De Luca, M., Dezani-Ciancaglini, M., and Yoshida, N. (2008b). Global Progress in Dynamically Interleaved Multiparty Sessions. In *CONCUR'08*, volume 5201 of *LNCS*, pages 418–433. Springer.

Bettini, L., De Nicola, R., and Loreti, M. (2008c). Implementing Session Centered Calculi. In *COORDINATION'08*, volume 5052 of *LNCS*, pages 17–32. Springer.

Bhargavan, K., Corin, R., Deniélou, P.-M., Fournet, C., and Leifer, J. J. (2009). Cryptographic Protocol Synthesis and Verification for Multiparty Sessions. In *CSF'09*, pages 124–140. IEEE Computer Society.

Bocchi, L., Honda, K., Tuosto, E., and Yoshida, N. (2010). A Theory of Design-by-Contract for Distributed Multiparty Interactions. In *CONCUR'10*, volume 6269 of *LNCS*, pages 162–176. Springer.

Bonelli, E. and Compagnoni, A. (2008). Multipoint Session Types for a Distributed Calculus. In *TGC'07*, volume 4912 of *LNCS*, pages 240–256. Springer.

Bonelli, E., Compagnoni, A., and Gunter, E. (2005). Correspondence Assertions for Process Synchronization in Concurrent Communications. *Journal of Functional Programming*, 15(2):219–248.

Bono, V., Messa, C., and Padovani, L. (2011). Typing Copyless Message Passing. In *ESOP'11*, volume 6602 of *LNCS*, pages 57–76. Springer.

Capecchi, S., Castellani, I., and Dezani-Ciancaglini, M. (2011). Information Flow Safety in Multiparty Sessions. In *EXPRESS'11*, volume 64 of *EPTCS*, pages 16–31.

Capecchi, S., Castellani, I., Dezani-Ciancaglini, M., and Rezk, T. (2010a). Session Types for Access and Information Flow Control. In *CONCUR'10*, volume 6269 of *LNCS*, pages 237–252. Springer.

Capecchi, S., Coppo, M., Dezani-Ciancaglini, M., Drossopoulou, S., and Giachino, E. (2009). Amalgamating Sessions and Methods in Object Oriented Languages with Generics. *Theoretical Computer Science*, 410:142–167.

Capecchi, S., Giachino, E., and Yoshida, N. (2010b). Global Escape in Multiparty Sessions. In *FSTTCS'10*, volume 8 of *LIPIcs*, pages 338–351. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

Carbone, M., Honda, K., and Yoshida, N. (2007). Structured Communication-Centred Programming for Web Services. In *ESOP'07*, volume 4421 of *LNCS*, pages 2–17. Springer.

Carbone, M., Honda, K., and Yoshida, N. (2008a). Multiparty Asynchronous Session Types. In *POPL'08*, pages 273–284. ACM Press.

Carbone, M., Honda, K., and Yoshida, N. (2008b). Structured Interactional Exceptions for Session Types. In *CONCUR'08*, volume 5201 of *LNCS*, pages 402–417. Springer.

Castagna, G., De Nicola, R., and Varacca, D. (2008). Semantic Subtyping for the π-calculus. *Theoretical Computer Science*, 398(1-3):217–242.

Castagna, G., Dezani-Ciancaglini, M., Giachino, E., and Padovani, L. (2009). Foundations of Session Types. In *PPDP'09*, pages 219–230. ACM Press.

Clarke, D., Noble, J., and Potter, J. (2001). Simple Ownership Types for Object Containment. In *ECOOP'01*, volume 2072 of *LNCS*, pages 53–76. Springer.

Coppo, M. and Dezani-Ciancaglini, M. (2009). Structured Communications with Concurrent Constraints. In *TGC'08*, volume 5474 of *LNCS*, pages 104–125. Springer.

Coppo, M., Dezani-Ciancaglini, M., and Yoshida, N. (2007). Asynchronous Session Types and Progress for Object-Oriented Languages. In *FMOODS'07*, volume 4468 of *LNCS*, pages 1–31. Springer.

Deniélou, P.-M. and Yoshida, N. (2010). Buffered Communication Analysis in Distributed Multiparty Sessions. In *CONCUR'10*, volume 6269 of *LNCS*, pages 343–357. Springer.

Dezani-Ciancaglini, M., de' Liguoro, U., and Yoshida, N. (2008). On Progress for Structured Communications. In *TGC'07*, volume 4912 of *LNCS*, pages 257–275. Springer.

Dezani-Ciancaglini, M., Drossopoulou, S., Giachino, E., and Yoshida, N. (2007). Bounded Session Types for Object-Oriented Languages. In *FMCO'06*, volume 4709 of *LNCS*, pages 207–245. Springer.

Dezani-Ciancaglini, M., Drossopoulou, S., Mostrous, D., and Yoshida, N. (2009). Session Types for Object-Oriented Languages. *Information and Computation*, 207(5):595–641.

Dezani-Ciancaglini, M., Mostrous, D., Yoshida, N., and Drossopoulou, S. (2006). Session Types for Object-Oriented Languages. In *ECOOP'06*, volume 4067 of *LNCS*, pages 328–352. Springer.

Dezani-Ciancaglini, M., Yoshida, N., Ahern, A., and Drossopoulou, S. (2005). A Distributed Object Oriented Language with Session Types. In *TGC'05*, volume 3705 of *LNCS*, pages 299–318. Springer.

Drossopoulou, S., Dezani-Ciancaglini, M., and Coppo, M. (2007). Amalgamating the Session Types and the Object Oriented Programming Paradigms. Presented at *MPOOL'07*.

Fähndrich, M., Aiken, M., Hawblitzel, C., Hodson, O., Hunt, G. C., Larus, J. R., and Levi, S. (2006). Language Support for Fast and Reliable Message-based Communication in Singularity OS. In *EuroSys2006*, ACM SIGOPS, pages 177–190. ACM Press.

Frisch, A., Castagna, G., and Benzaken, V. (2008). Semantic Subtyping: Dealing Set-theoretically with Function, Union, Intersection, and Negation Types. *Journal of the ACM*, 55(4):1–64.

Gapeyev, V. and Pierce, B. C. (2003). Regular Object Types. In *ECOOP'03*, volume 2743 of *LNCS*, pages 151–175. Springer.

Garralda, P., Compagnoni, A., and Dezani-Ciancaglini, M. (2006). BASS: Boxed Ambients with Safe Sessions. In *PPDP'06*, pages 61–72. ACM Press.

Gay, S. (2008). Bounded Polymorphism in Session Types. *Mathematical Structures in Computer Science*, 18(5):895–930.

Gay, S. and Hole, M. (2005). Subtyping for Session Types in the Pi-Calculus. *Acta Informatica*, 42(2/3):191–225.

Gay, S., Vasconcelos, V. T., and Ravara, A. (2003). Session Types for Inter-Process Communication. TR 2003–133, Department of Computing, University of Glasgow.

Gay, S. J., Vasconcelos, V. T., Ravara, A., Gesbert, N., and Caldeira, A. Z. (2010). Modular Session Types for Distributed Object-oriented Programming. In *POPL'10*, pages 299–312. ACM Press.

Giachino, E. (2009). *Session Types: Semantic Foundations and Object-Oriented Applications*. PhD thesis, Università degli Studi di Torino – Université Paris 7.

Honda, K. (1993). Types for Dyadic Interaction. In *CONCUR'93*, volume 715 of *LNCS*, pages 509–523. Springer.

Honda, K., Mostrous, D., and Yoshida, N. (2009). Global Principal Typing in Partially Commutative Asynchronous Sessions. In *ESOP'09*, volume 5502 of *LNCS*, pages 316–332. Springer.

Honda, K., Vasconcelos, V. T., and Kubo, M. (1998). Language Primitives and Type Disciplines for Structured Communication-based Programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 22–138. Springer.

Honda, K., Yoshida, N., and Carbone, M. (2007). Web Services, Mobile Processes and Types. *EATCS Bulletin*, 91:160–188.

Hu, R., Kouzapas, D., Pernet, O., Yoshida, N., and Honda, K. (2010). Type-Safe Eventful Sessions in Java. In *ECOOP'10*, volume 6183 of *LNCS*, pages 329–353. Springer.

Hu, R., Yoshida, N., and Honda, K. (2008). Session-Based Distributed Programming in Java. In *ECOOP'08*, volume 5142 of *LNCS*, pages 516–541. Springer.

Igarashi, A. and Nagira, H. (2007). Union Types for Object Oriented Programming. *Journal of Object Technology*, 6(2):31–52.

Igarashi, A., Pierce, B. C., and Wadler, P. (2001). Featherweight Java: a Minimal Core Calculus for Java and GJ. *ACM TOPLAS*, 23(3):396–450.

Lieberman, H. (1986). Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems. In *OOPSLA'86*, volume 21(11), pages 214–223. ACM Press.

Mostrous, D. and Yoshida, N. (2007). Two Sessions Typing Systems for Higher-Order Mobile Processes. In *TLCA'07*, volume 4583 of *LNCS*, pages 321–335. Springer.

Mostrous, D. and Yoshida, N. (2009). Session-Based Communication Optimisation for Higher-Order Mobile Processes. In *TLCA'09*, volume 5608 of *LNCS*, pages 203–218. Springer.

Pierce, B. C. (2002). *Types and Programming Languages*. MIT Press.

Sparkes, S. (2006). Conversation with Steve Ross-Talbot. *ACM Queue*, 4(2):14–23.

Takeuchi, K., Honda, K., and Kubo, M. (1994). An Interaction-based Language and its Typing System. In *PARLE'94*, volume 817 of *LNCS*, pages 398–413. Springer.

Vallecillo, A., Vasconcelos, V. T., and Ravara, A. (2002). Typing the Behavior of Objects and Components using Session Types. In *FOCLASA'02*, volume 68(3) of *ENTCS*, pages 439–456. Elsevier.

Vasconcelos, V. T., Gay, S., and Ravara, A. (2006). Typechecking a Multithreaded Functional Language with Session Types. *Theorical Computer Science*, 368(1-2):64–87.

Web Services Choreography Working Group (2002). Web Services Choreography Description Language. http://www.w3.org/2002/ws/chor/.

Yoshida, N., Deniélou, P.-M., Bejleri, A., and Hu, R. (2010). Parameterised Multiparty Session Types. In *FOSSACS'10*, volume 6014 of *LNCS*, pages 128–145. Springer.

Yoshida, N. and Vasconcelos, V. T. (2007). Language Primitives and Type Discipline for Structured Communication-Based Programming Revisited: Two Systems for Higher-Order Session Communication. In *SecRet'06*, volume 171(4) of *ENTCS*, pages 73–93. Elsevier.